freefloat

# Freefloat Link
version 6.0

## User's Manual

*Contacting Freefloat* ..... *5*

*What is Freefloat Link?* ..... *6*

Features ..... 7

**Supported Platforms** ..... **7**

*Getting Started* ..... *8*

**Starting Freefloat Link for the First Time** ..... **8**

**Testing with a Serial Device** ..... **10**

**Testing with a Hot Key** ..... **11**

**The Wizard** ..... **13**

*Freefloat Link Settings* ..... *17*

**Main Window** ..... **17**

**Settings Dialog** ..... **18**

The Communication Tab ..... 18
Keyboard Tab ..... 20
Key Settings ..... 21
Hot Keys ..... 26
Keyboard Captures ..... 30
Record Key Sequence ..... 36
Data Editing Tab ..... 40
Misc Tab ..... 42
Set Password Dialog ..... 43
Where Are My Settings? ..... 43

*Freefloat Link Scripting* ..... *44*

**Overview** ..... **44**

**The Lua Language** ..... **45**

**Script Events** ..... **46**

**Event Methods** ..... **47**

onStart() ..... 47
onEnd() ..... 47
onData( data, length ) ..... 48
onHotKey( name ) ..... 49
onKeyboardCapture( name, data ) ..... 50
onExternalData( data, length ) ..... 50
onTimer() ..... 52
onCTS( status ) ..... 53
Script Methods ..... 54
Output/User Feedback ..... 56
beep( frequency, duration ) ..... 56
blinkIcon( icon, duration ) ..... 57
log( filename, message ) ..... 58
messageBox( title, message, type ) ..... 59
playSound( filename, options ) ..... 61
playSystemSound( systemEvent, options ) ..... 62
send( data ) ..... 63

sendSerialData( data, length ) 64
sendSubscriberData( data, length ) 65
Windows 66
enumWindows( handle ) 66
findWindow( title, class ) 67
getForegroundWindow() 68
getWindowClass( handle ) 69
getWindowText( handle ) 70
setForegroundWindow( handle ) 71
setWindowText( handle, text ) 72
windowOperation( handle, operation ) 73
Clipboard 74
getClipboardData() 74
setClipboardData( text ) 75
Application Launch 76
closeAppHandle( handle ) 76
isAppRunning( handle ) 77
run( program, argument, delay ) 78
Serial Port 79
closePort() 79
getDTR() 80
getRTS() 81
openPort() 82
setDTR( status ) 83
setRTS( status ) 84
Miscellaneous 85
ean128( data, strict ) 85
exit() 86
exitWindows( options ) 87
getProfile() 88
getTickCount() 89
lockWorkStation() 90
setProfile( profile ) 91
setTimer( interval ) 92
sleep( duration ) 93
***Notification Area Icon 94***
***Migration guide WLinq 3.x to Freefloat Link 95***
**Duplicate String Filter 96**
**Case Setting 97**
**Character Translation 98**
**Send Pre- and Postfix Keys 99**
**Lock Output Window 99**
**Initialization String 100**
**Filter Unknown Data Strings 101**
**Input Data Replacements 102**
**Criteria 103**
**Data Format Output 105**

*Appendix A, Support for Thin Clients, Java Applications, and Flash Applications ..... 111*
*Appendix B. Lua Copyright ..... 112*
*Appendix C. Version History ..... 113*

Copyright © 2008–2012 Freefloat

## Contacting Freefloat

***Office***
Freefloat
P.O. Box 13101
S-402 52  Gothenburg
SWEDEN

***Support and Sales Information***

| | |
|---|---|
| Support | http://support.freefloat.com/ |
| Web | http://www.freefloat.com/ |
| E-mail | info@freefloat.com |

## What is Freefloat Link?

Freefloat Link is a flexible and powerful wedge software which enables you to capture data from serial and USB-HID devices connected to your PC, filter and modify this data, and then enter this information into a receiving application.

Freefloat Link can be used to integrate for example barcode reading into an existing application without the need to adapt the target application.

Instead of using a large amount of functions and features, which no matter how advanced they are, never achieves true flexibility, the design philosophy of Freefloat Link is to have only a basic set of settings and expose all methods to a dynamic script language.

For simple demands, there is a wizard that can be used to create data editing rules. This manual has a lot of examples for common uses. This enables users without programming skills to accomplish the desired task. For more advanced solutions, basic programming skills are needed.

## Features

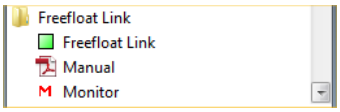
- ❑ Event based architecture
- ❑ The dynamic script language Lua has been embedded which makes Freefloat Link very powerful, flexible and extensible
- ❑ Multiple input sources
  - Keyboard capture (USB-HID devices)
  - Hot keys
  - Serial data (RS-232)
  - Serial handshake signal changes
  - External application data
  - Start, end, and timer events
- ❑ Multiple instances of Freefloat Link can be used at the same time
- ❑ Monitor program for real-time inspection of events and output, easing development of a solution or as an aid in troubleshooting
- ❑ Detailed log file
- ❑ Powerful key sequence recorder used for:
  - key definitions/macros
  - hot keys
  - keyboard captures
- ❑ Polarity of serial output signals (DTR and RTS) can be set from a script
- ❑ Clipboard methods for faster transfer of data to applications
- ❑ Programs can be launched, synchronously or asynchronously
- ❑ Default key definitions and ASCII/Unicode data output are insensitive to keyboard locale
- ❑ Script user feedback through sounds (wave files or internal speaker) and message boxes
- ❑ GS1-128 (UCC/EAN-128) support
- ❑ Custom application interface for tight high-speed integration with Freefloat Link

## Supported Platforms

Windows 2000
Windows XP/XPe
Windows Vista
Windows 7, 32 and 64-bit

## Getting Started

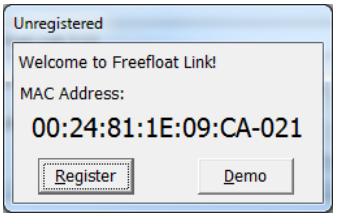
When you install Freefloat Link, shortcuts are created on the Start menu.




Here you can start Freefloat Link, this manual, or the Monitor program.

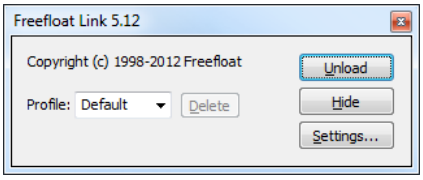
### Starting Freefloat Link for the First Time

Navigate to the *Freefloat Link* folder on the Start menu and then click on Freefloat Link. Freefloat Link starts and the main window and the licensing dialog appear:



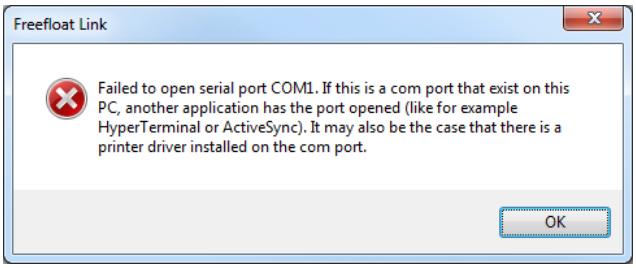

Click **Register** and enter your license key supplied to you by your reseller. Or click **Demo** to run Freefloat Link in demo mode. The demo mode allows for twenty events to be handled. Freefloat Link can be restarted to get twenty more events.

When Freefloat Link is licensed or you click the **Demo** button in the **Unregistered Dialog** the main window is displayed:




Please note that Freefloat Link adds an icon to the notification area (the left-most one in the picture below).

By default, Freefloat Link tries to open the serial port COM1 with the settings 9600 bps, 8 data bits, no parity and one stop bit. If your PC does not have a serial port called COM1 or if COM1 has been opened by another application, Freefloat Link displays the following error message:


Freefloat Link
Failed to open serial port COM1. If this is a com port that exist on this PC, another application has the port opened (like for example HyperTerminal or ActiveSync). It may also be the case that there is a printer driver installed on the com port.
OK

When Freefloat Link fails to open a COM port the notification icon turns red:

When this happens, Freefloat Link 's entire script engine is turned off. If you plan to not use any COM port at all, you need to click on the **Settings** button in the main window, select the **Communication** tab, and then remove the check from **Enable**.

For a list of possible icon colors and when they are used, please see the topic **Notification Area Icon**.

## Testing with a Serial Device

For this test to work, you need a serial device (for example a barcode scanner) that uses the settings 9600 bps, 8 data bits, no parity and one stop bit. If your serial device uses other settings, please refer to later sections in this manual for information on how to adjust the settings in Freefloat Link to match those in your device.

Most serial devices terminates data strings with a carriage return (ASCII code 13). This is also what Freefloat Link expects as default.

Connect your serial device to COM1, start Notepad and activate it. Read a barcode or take the action that makes your serial device send a string. If Freefloat Link receives serial data the notification icon turns green:

If the icon does not turn green, check the following:

- ❑ The serial settings in Freefloat Link matches those in your device
- ❑ The device requires one or both of the handshake signals (DTR/RTS) to be set in a specific way. Default they are both set to high
- ❑ Check that the serial device is connected to COM1 and turned on
- ❑ Make sure the device's cable is a standard RS-232 cable
- ❑ You don't have the **Settings** dialog open

If the icon turns green but nothing is entered into Notepad, your serial device probably uses a different terminator. Do the following:

- ❑ Try clicking on **Settings**, select the **Data Editing** tab, select **Timeout** as the **Data String Termination** option and then click **OK**.

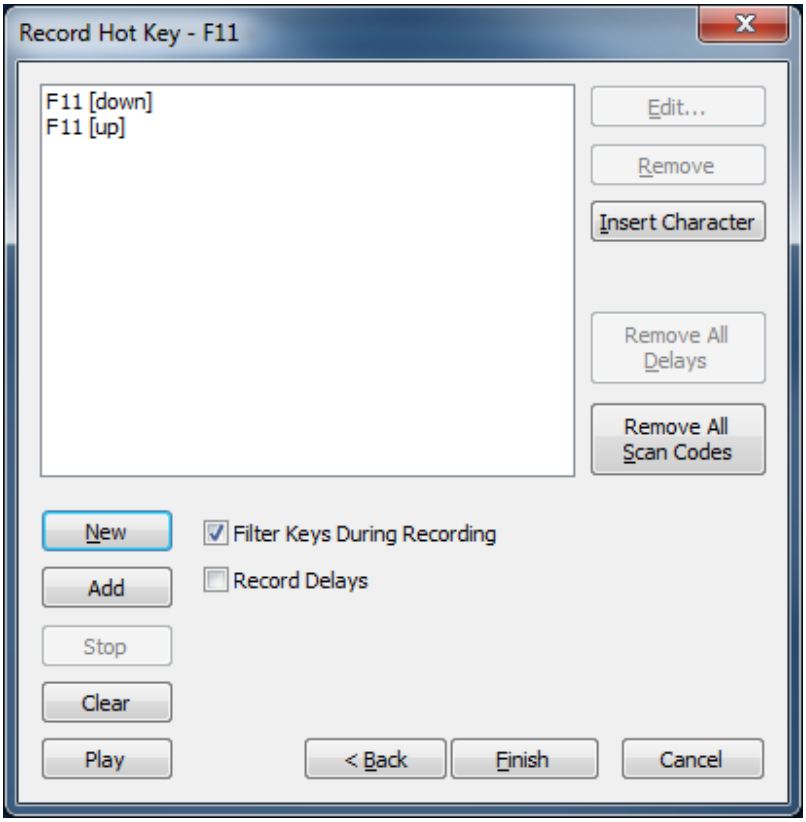
The data string will now be terminated after a small period of time has elapsed. You may also use the Monitor program to troubleshoot data string terminator problems.

## Testing with a Hot Key

If you don't have a serial device available or you do not plan to use one you could try to define a hot key as your first exercise.

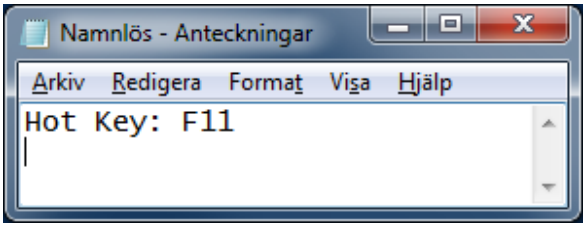
Click on the **Settings** button in the main window, then click the **Communication** tab, and then remove the check from **Enable** to turn off serial communication. Then click on the **Keyboard** tab, and click on the **Hot Keys** button. Click **Add...** and enter a name for the hot key. Name it **F11**. The **Record Hot Key** dialog is displayed:

Click on the **New** button to start recording your hot key. Press the F11 key on your keyboard and then click the **Stop** button. The result should look like this:




Click **Finish**, click **OK**, and click **OK**.

Activate Notepad, and then press the key F11 on your keyboard. The text "Hot Key: F11" should now be entered into Notepad.

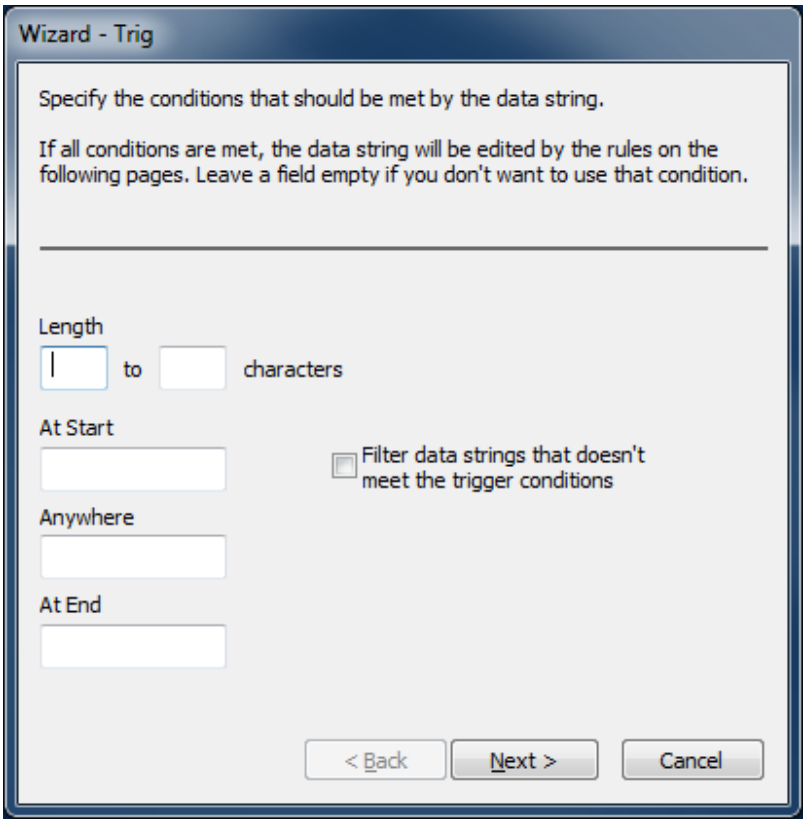


## The Wizard

The wizard enables you to define a simple setup that matches and modifies a data string (for example a barcode). It is only intended to be used for very simple tasks. For more advanced tasks, you need to use the scripting language.

The wizard consist of four different parts, **Trig**, **Strip**, **Replace**, and **Add**.

You start the wizard by clicking on the button **Wizard...** in the **Data Editing** tab of the **Settings** dialog.

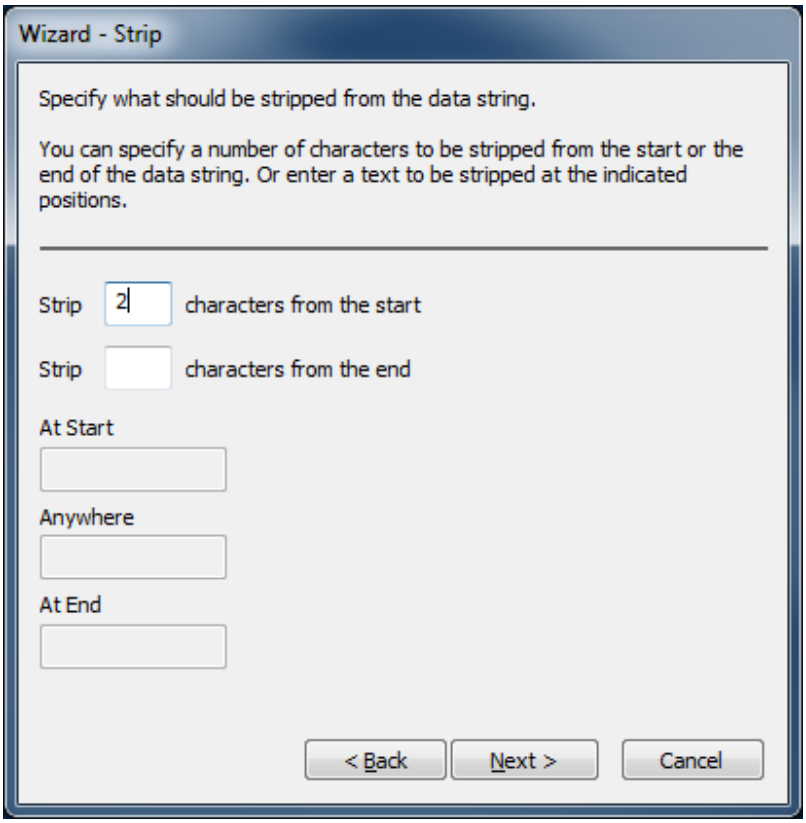

Wizard - Trig
Specify the conditions that should be met by the data string.
If all conditions are met, the data string will be edited by the rules on the following pages. Leave a field empty if you don't want to use that condition.
Length
to
characters
At Start
Filter data strings that doesn't meet the trigger conditions
Anywhere
At End
< Back
Next >
Cancel

In this dialog you can specify zero or more conditions that should be met for a data string before it is edited by the wizard rules. If a data string doesn't meet the conditions and the check box **Filter Data Strings** is checked, the data string is filtered. If the check box is cleared, the data string is sent to the receiving application unmodified.

If you specify a length condition, any data strings shorter or longer will not be modified by the wizard rules.

You can enter texts that should be present in the data string. Entering 00 in the **At Start** field will check for 00 in the beginning of the data string. Then any string that starts with something else will not meet the condition.

Click the **Next > button** when you are done.


Wizard - Strip
Specify what should be stripped from the data string.
You can specify a number of characters to be stripped from the start or the end of the data string. Or enter a text to be stripped at the indicated positions.
Strip
2
characters from the start
Strip
characters from the end
At Start
Anywhere
At End
< Back
Next >
Cancel

Here you can choose between two options:

- ❑ Stripping a certain number of characters at the start and/or the end of the data string

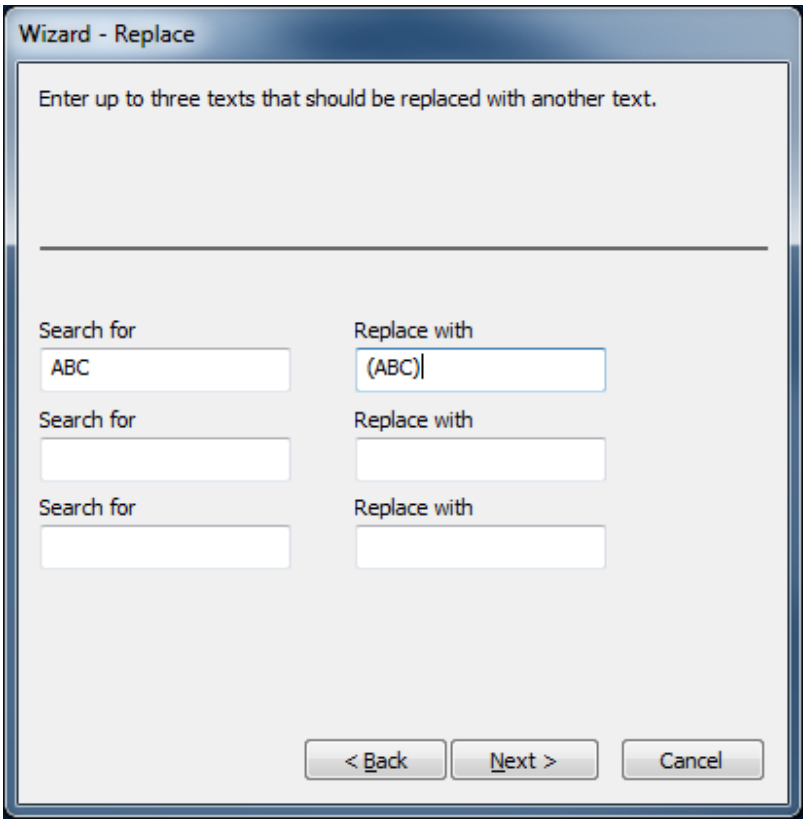
  or

- ❑ Removing one or more texts at the indicated positions of the data string

For example, if you specify 00 in the field **At Start** and the data string is 0012345, the resulting output will be 12345. However, if the data string is 9912345, nothing will be removed from the start of the data string.

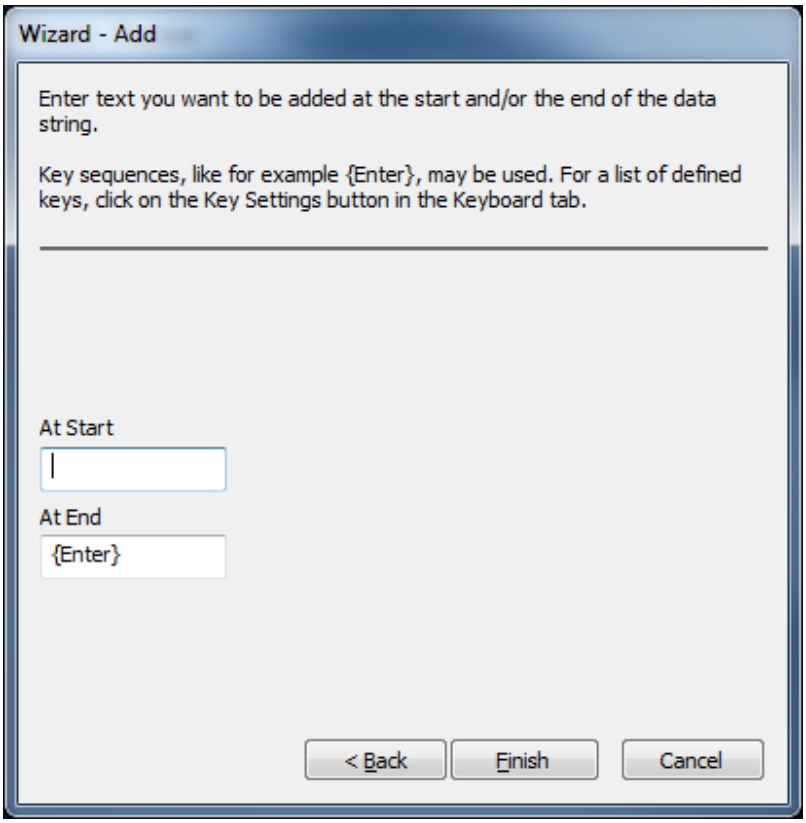
If you leave all fields empty, no text will be stripped from the data string.

Click the **Next >** button when you are done.




Here you can enter up to three text replacements that should be done. Enter the text you want to replace in one of the **Search for** fields, and then enter the text you want to replace it with in the corresponding **Replace with** field.

Click the **Next >** button when you are done.


Wizard - Add
Enter text you want to be added at the start and/or the end of the data string.
Key sequences, like for example {Enter}, may be used. For a list of defined keys, click on the Key Settings button in the Keyboard tab.
At Start
At End
{Enter}
< Back
Finish
Cancel

In this last dialog you can enter texts that should be added to the data string at the start or the end.

When you are done, click on the **Finish** button. When you do, you need to confirm that you want to create the wizard script.

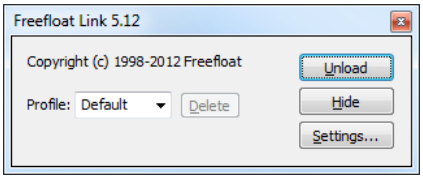
The wizard script is a Lua script and can be modified manually afterwards if you for example need to add more advanced conditions or modifications. Please note that the wizard script is overwritten each time you run the wizard.

## Freefloat Link Settings

This section explains the settings in Freefloat Link. For information about where the settings are actually stored, please refer to the section **Where Are My Settings?** later in this section.

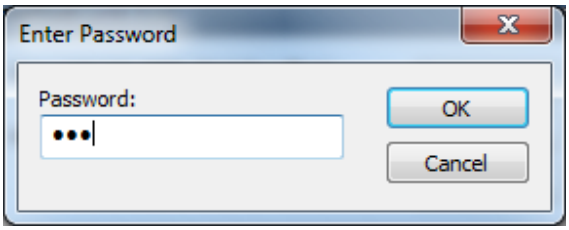
### Main Window

From the main window you can control Freefloat Link, modify the settings and so on.




| Action | Description |
|---|---|
| **Unload** | Exits Freefloat Link. |
| **Hide**, Clicking or pressing Esc | Hides the main window.<br><br>Click the notification area icon to show the main window if it is hidden. |
| **Settings** | Displays the **Settings** dialog. |
| Click in the **Profile** edit box; enter a new profile name and the click on **Settings**. | Creates a new profile with the name you entered. |
| **Delete** | Deletes the selected profile. **Note!** You can't delete the profile called Default. |

**Note!** If a password has been set, clicking **Unload** or **Settings** will display the following dialog:

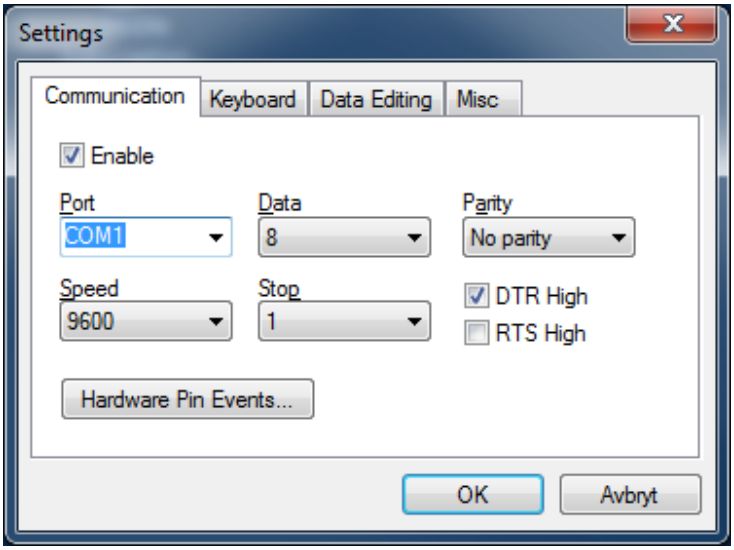



You must enter the correct password to display the **Settings** dialog or to exit Freefloat Link. This feature blocks users from messing with the configuration.

## Settings Dialog

In the main window, click on the **Settings** button to display the **Settings** dialog. The settings dialog consists of a number of tabs for different settings.

## The Communication Tab



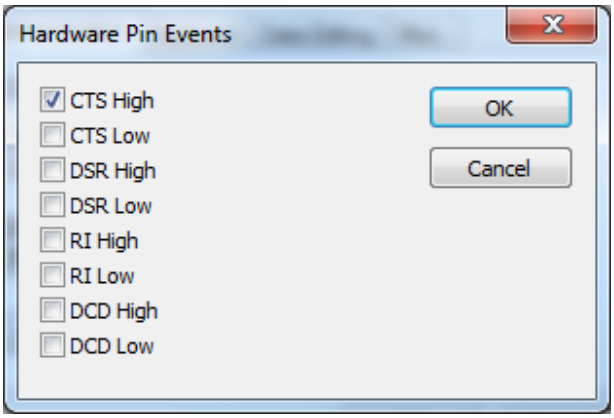

The **Port**, **Speed**, **Data**, **Stop**, and **Parity** are the settings for the serial port for Freefloat Link to open and use.

The ports COM1 to COM256 are supported. If you have a serial port that has a special name, for example BSP2:, you can enter the name in the **Port** edit box.

When **DTR High** or **RTS High** is checked, the corresponding handshake signal of the serial port will be set. Some serial devices require these to be set to enable communication.

**Note!** The DTR and RTS handshake signal can be controlled dynamically from a script.

Clicking on the **Hardware Pin Events...** button will display the following dialog:

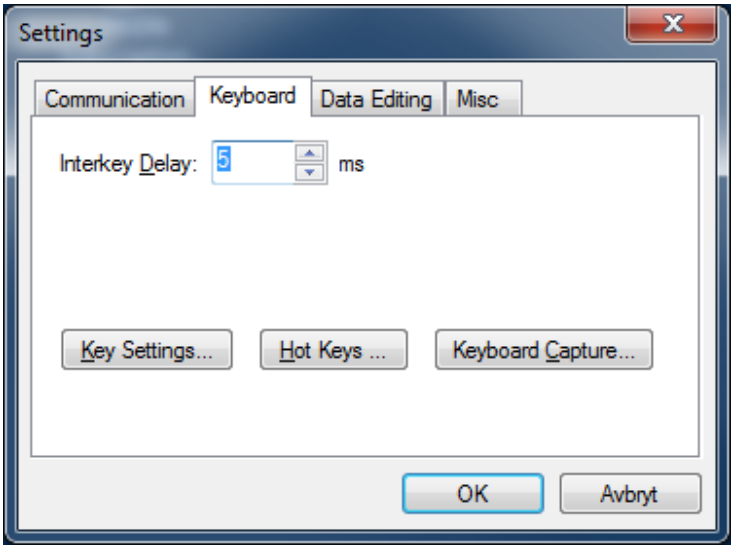



In a serial port there are four incoming signals called CTS, DSR, RI, and DCD. Freefloat Link can be set up to monitor these signals and generate an event when a signal is changed. An event can be generated when the signal goes high and/or when it goes low.

If the event is enabled in this dialog, a corresponding method in the script will be called. The default implementations of these methods send the signal name and its status (high or low):

```
function convertSignal( status )
  if status then
    return "High"
  else
    return "Low"
  end
end

function onCTS( status )
  app.send( "CTS " .. convertSignal( status ) .. "{Enter}" )
end

function onDSR( status )
  app.send( "DSR " .. convertSignal( status ) .. "{Enter}" )
end

function onRI( status )
  app.send( "RI " .. convertSignal( status ) .. "{Enter}" )
end

function onDCD( status )
  app.send( "DCD " .. convertSignal( status ) .. "{Enter}" )
end
```

## Keyboard Tab

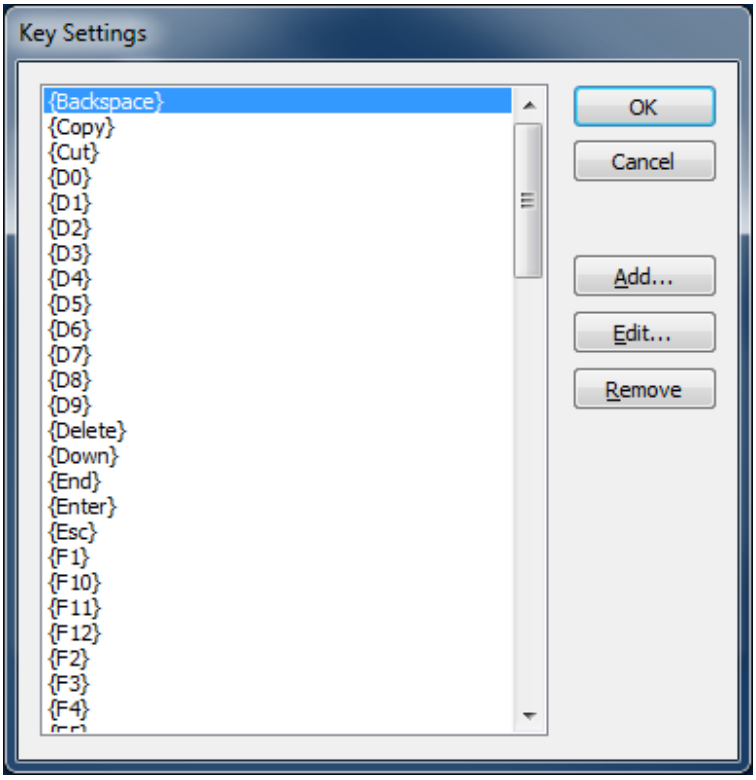



The **Interkey Delay** specifies the delay that should be used between each key press when simulating keyboard data to an application. For example, Microsoft's Terminal Services client in full screen mode looses key presses if this is set to zero. Note that this is a global delay. A recorded key sequence may contain additional delays between key presses.

## Key Settings

A key definition is a named key sequence. Key definitions are referred to in an expression passed to the **app.send()** method which replays the key presses recorded in the key definition.

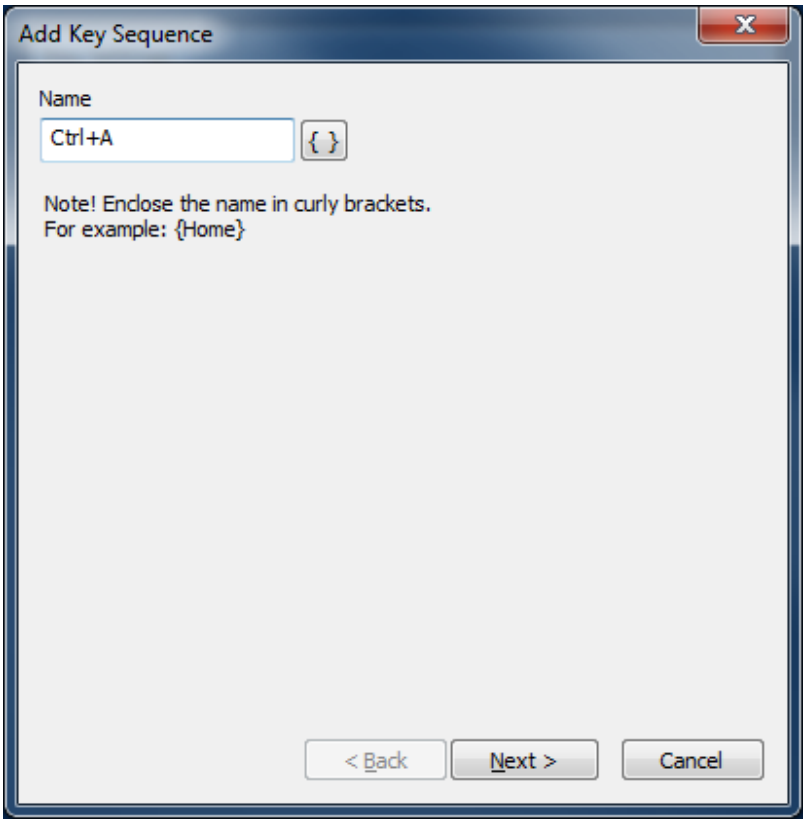
When clicking the **Key Settings...** button in the **Keyboard** tab the following dialog is displayed:



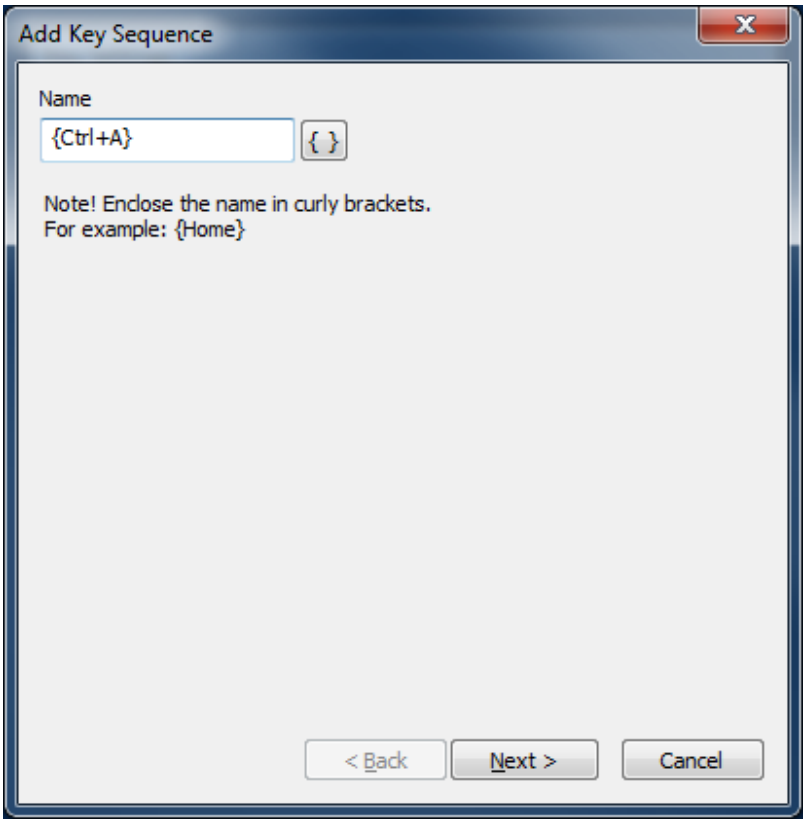

By default, Freefloat Link defines many of the standard keys on the keyboard. You can add, edit and remove a key definition.

For example, let's say you want to add the key sequence Ctrl+A: Click on **Add...** in the dialog above.

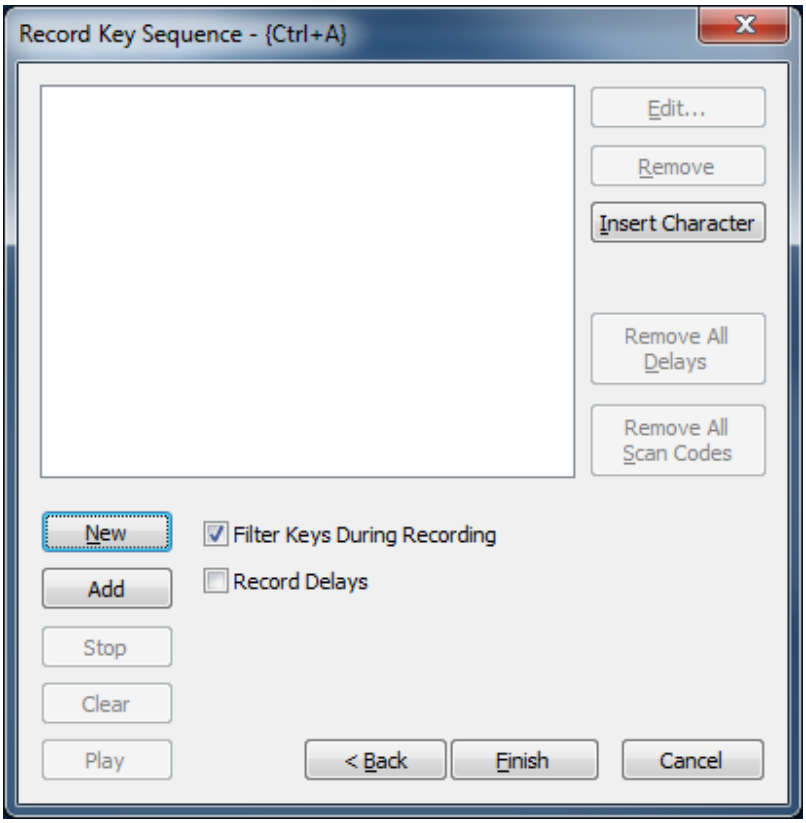
Enter the name Ctrl+A in the **Name** edit box:




Then click on the { } button.



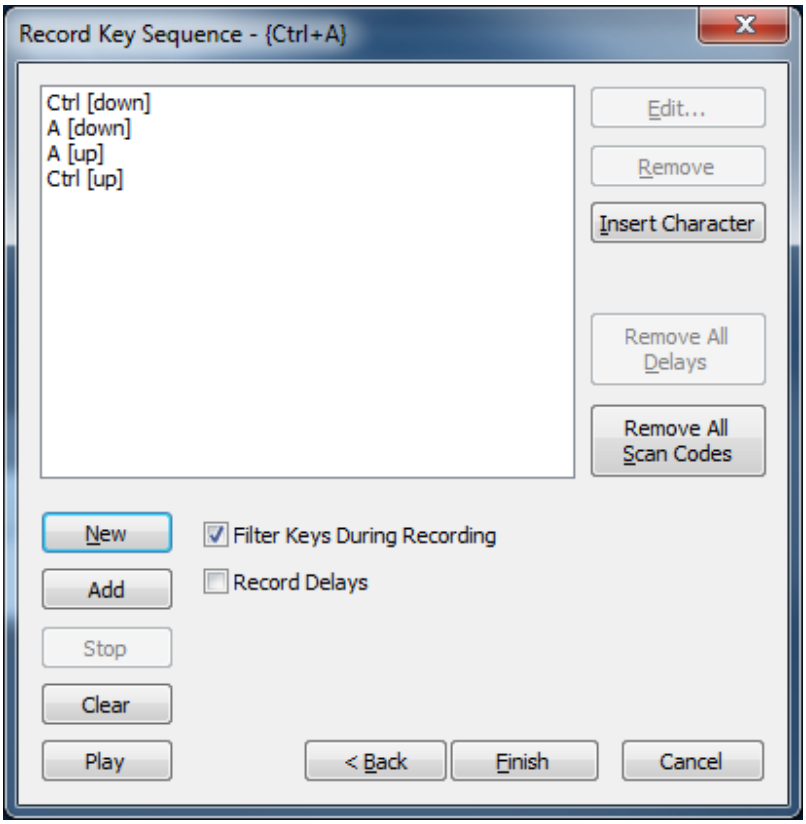

All key names must be enclosed in { and }. Click **Next >**.




Click **New** to start recording your sequence and then press and hold the Ctrl key on the keyboard and then press the A key. Release both keys. Click on the **Stop** button.

***Note!*** *The* ***Record Key Sequence*** *is a dialog common for key definitions, hot keys, and keyboard captures. It is explained in detail in a separate section later on.*

The dialog should now look like this:




Click **Finish** to complete the key definition.

Key definitions are used with the method **app.send()** from a script. For example:

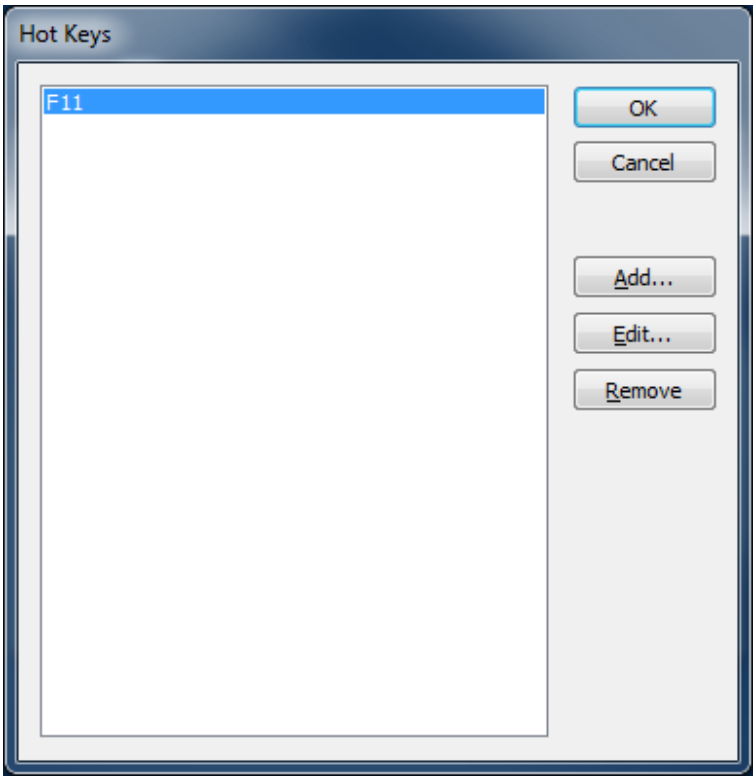
```
function onData( data, length )
  app.send( data .. "{Ctrl+A}" )
end
```

For more information about **app.send()**, see the topic **Freefloat Link Scripting**.

## Hot Keys

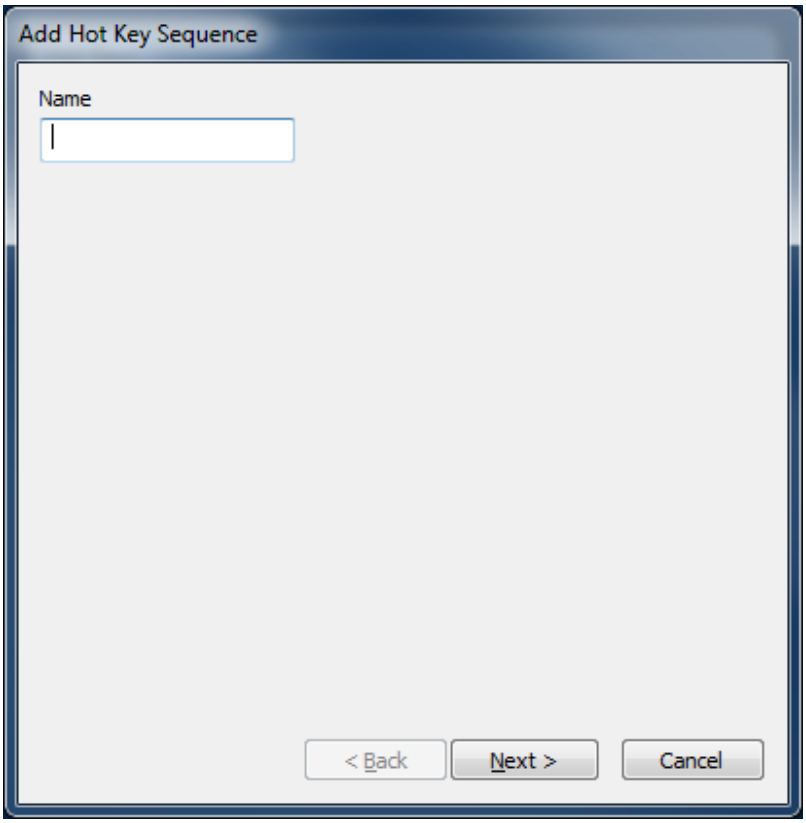
A hot key is a key sequence that when pressed causes the script method **onHotKey()** to be called.

When clicking the **Hot Keys...** button in the **Keyboard** tab the following dialog is displayed:



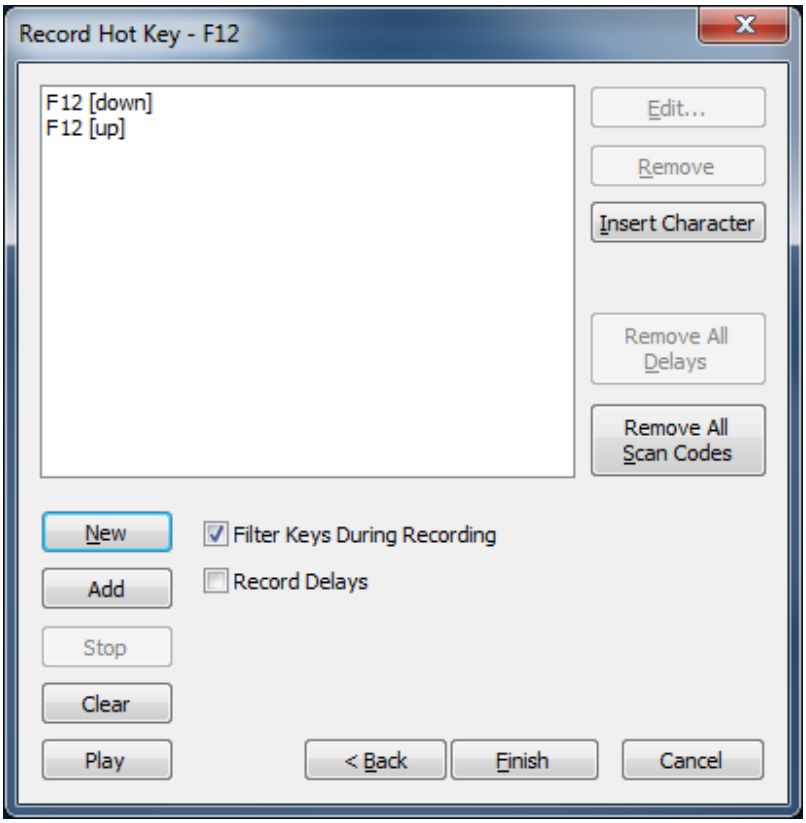

By default, Freefloat Link does not contain any hot key definitions. In the above dialog you can add, edit and remove hot keys.

To add a hot key, click on the **Add...** button.




Enter a name for the hot key and click **Next**.

Click on **New**, and then press the key or a combination of keys to define your hot key. For example, press F12. Then click **Stop**.




Please note that the hot key sequence above consists of both the F12 down event and the F12 up event. You can remove the up event from the key sequence to make the hot key feel more responsive but remember that, if you do so, the F12 up event will be passed to the application. This is not a problem because most application reacts to key presses on the down event but it may cause problems in some special circumstances.

Click the **Finish** button to complete the hot key definition.

**Note!** A hot key sequence is global in Windows. In the above case, F12 is filtered out from all applications while Freefloat Link is running.

The default implementation of the **onHotKey()** method looks like this:

```
function onHotKey( name )
  app.send( "Hot Key: " .. name .. "{Enter}" )
end
```

It simply enters "Hot Key: <hot key name>". Of course, hot keys can be made to do more useful things.

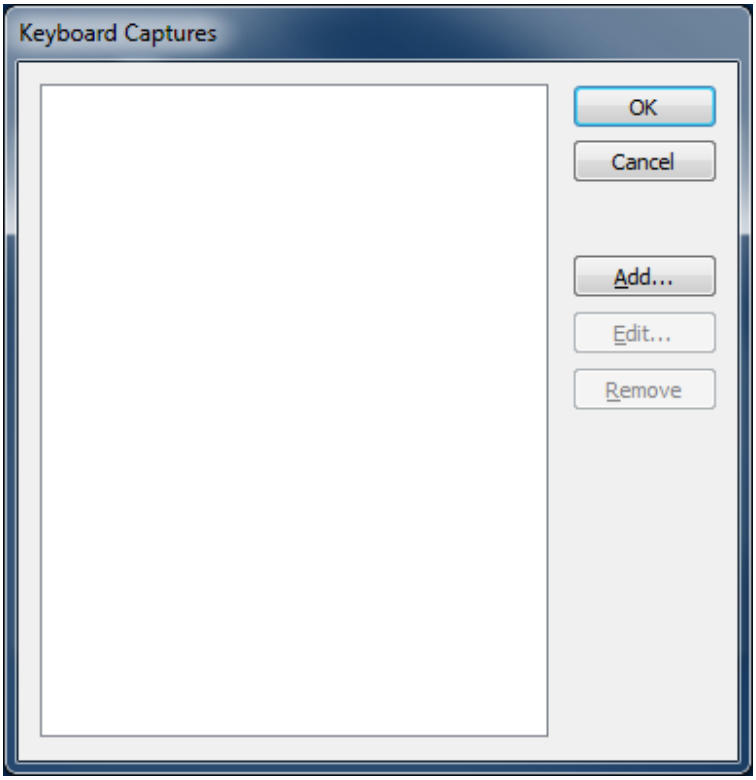
## Keyboard Captures

A keyboard capture consists of a name, a prefix key sequence and a suffix key sequence. It is used to capture data strings from a USB-HID device (for example a USB connected barcode scanner).

For this to work, the USB device needs to be configure to send a special key sequence before the data string and a key sequence that terminates the data string. Please note that the prefix sequence should be chosen with care. All the keys in the keyboard capture's prefix sequence will be filtered from regular keyboard input until a mismatch is found.

When a data string is captured by a keyboard capture, the script method **onKeyboardCapture()** is called.

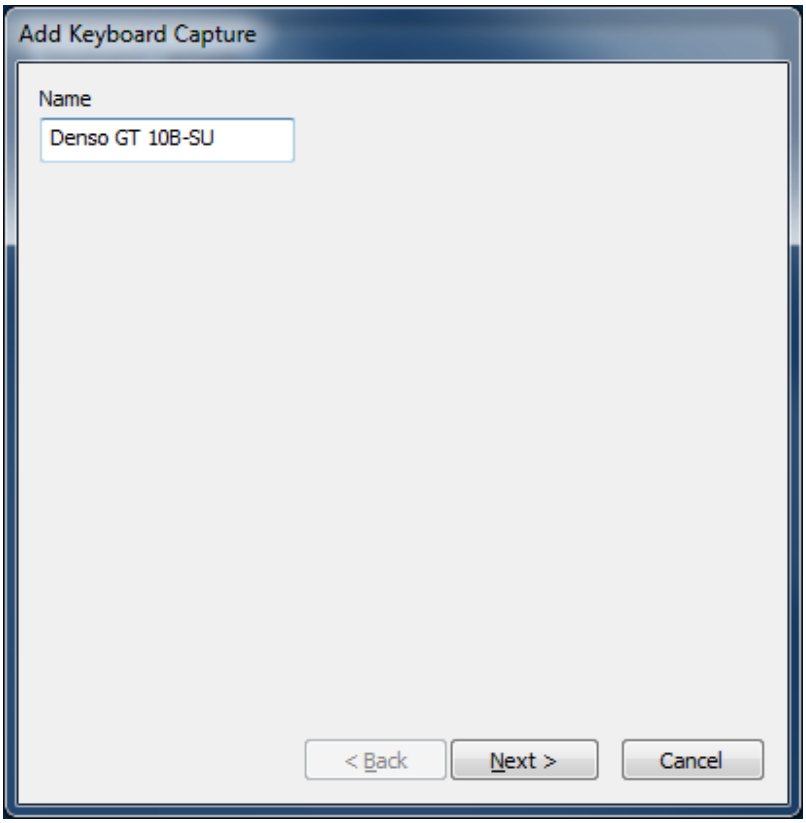
When clicking the **Keyboard Capture...** button in the **Keyboard** tab the following dialog is displayed:



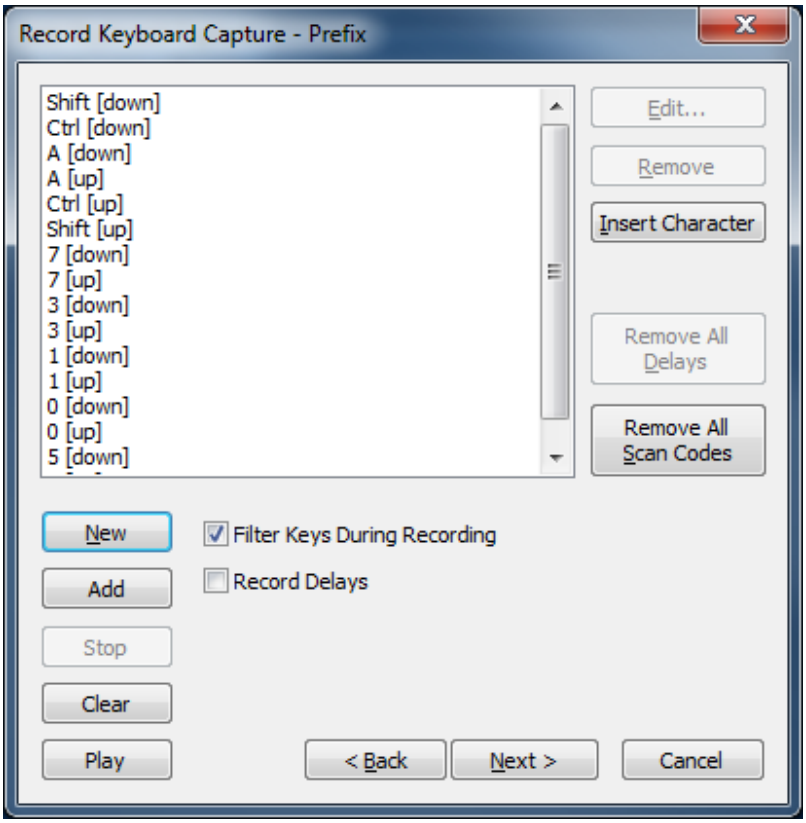

By default, Freefloat Link does not contain any keyboard captures. In the above dialog you can add, edit and remove keyboard captures.

To add a keyboard capture, click on the **Add...** button.

The following example shows how to define a keyboard capture for a specific USB-HID scanner. The scanner has been setup to send the key sequence Shift down, Ctrl down, A down, A up, Ctrl up, and Shift up.

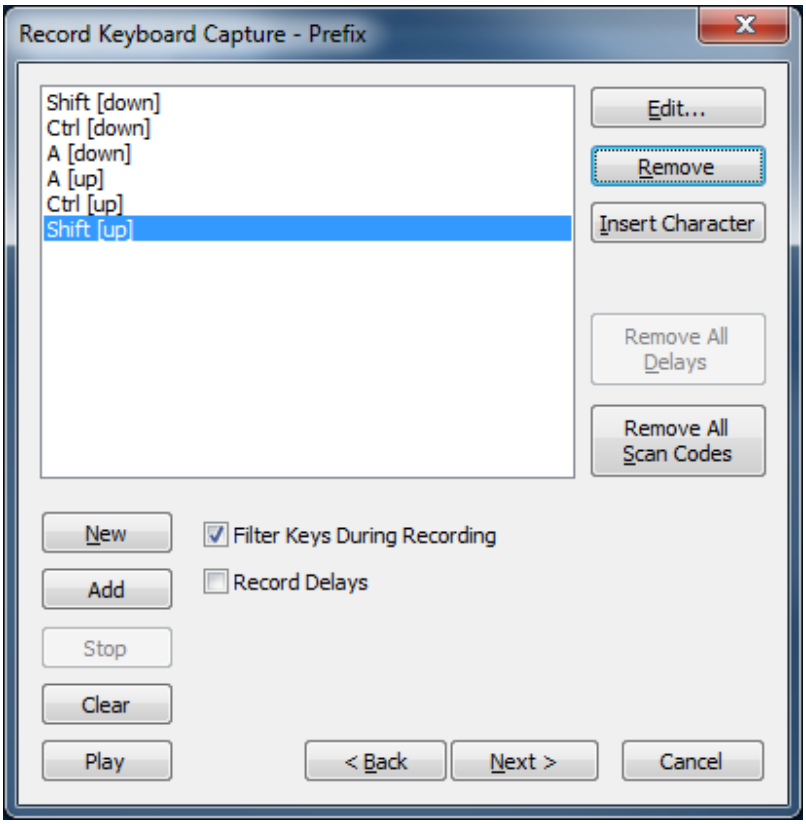



Enter the name then click the **Next** button. The dialog **Record Keyboard Capture - Prefix** is displayed. Click on the **New** button, read a barcode with the USB scanner and then click the **Stop** button.

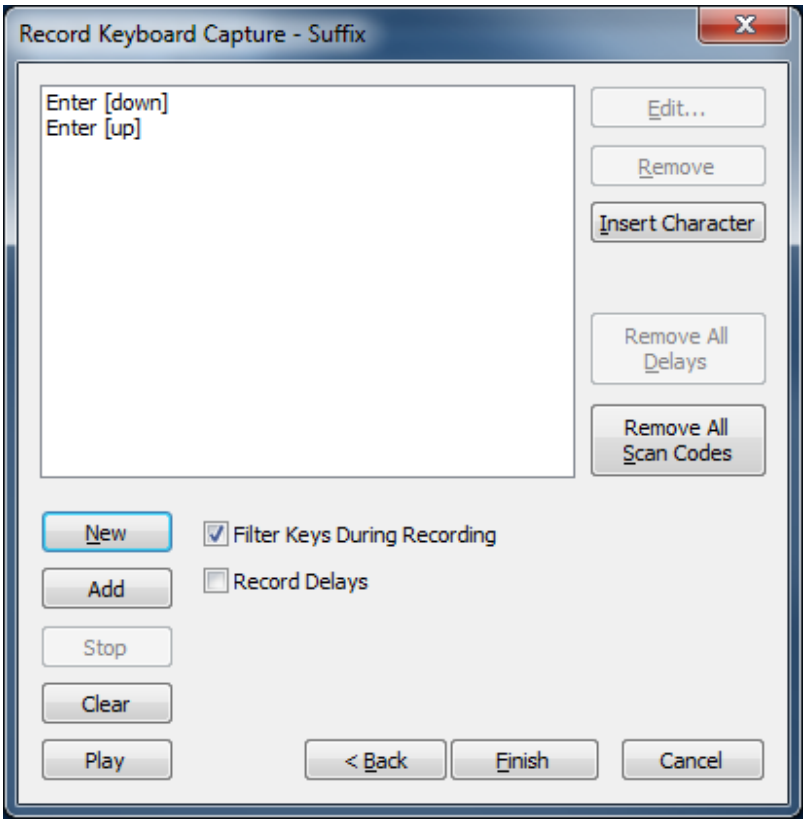



Note that all the keys the scanner generates when reading a barcode gets recorded: prefix, barcode data, and suffix. In the dialog above, the line directly after the prefix has been selected. You need to click on the **Remove** button repeatedly to remove the key events for the barcode data and suffix from the list.

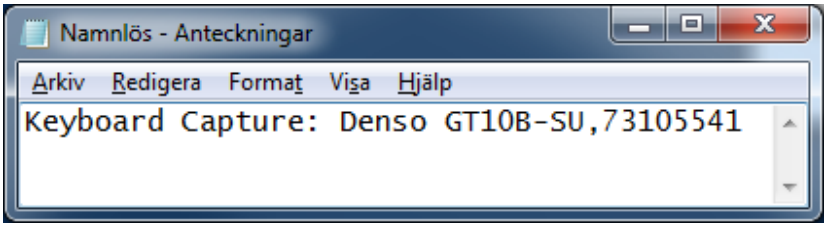
Left in the list is a key sequence that only contains the key events for the prefix:



Click **Next** and repeat the recording, but this time remove all the key events except the suffix:




Click **Finish** to complete the keyboard capture definition. If you have a USB-HID barcode scanner which you have adapted the above example to, you can now click **OK** until the main window of Freefloat Link is displayed, set the focus to Notepad and test the keyboard capture. If it works, something like this should be displayed:




If you only get the barcode contents, the keyboard capture is not working. This is probably because a mistake was made in the setup of the scanner or the prefix sequence.

If you don't get anything and the keyboard seems to have stopped working, then the suffix sequence is probably wrong.

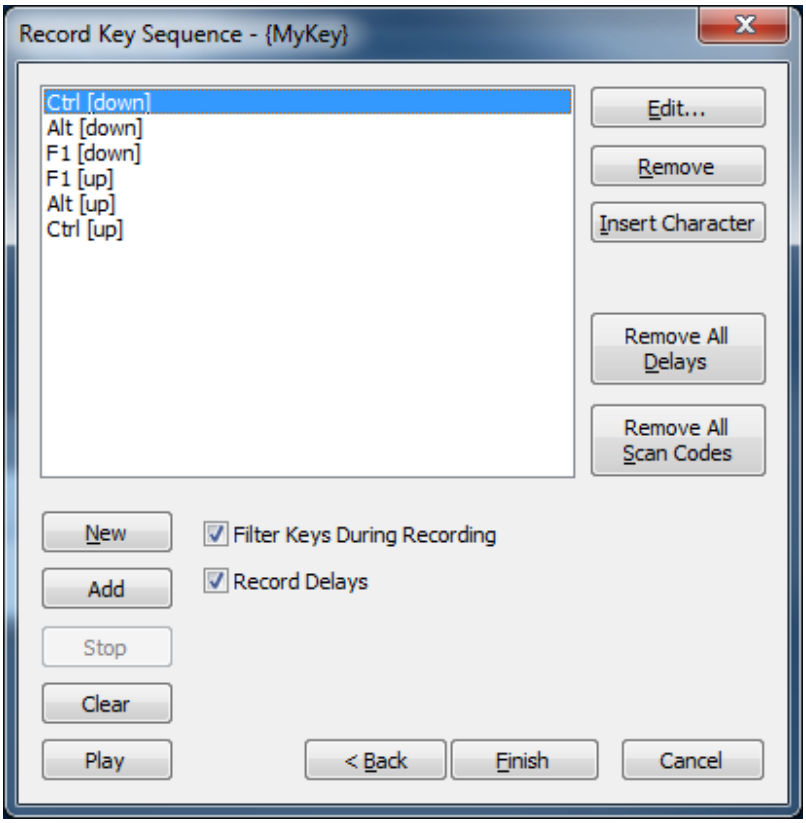
The default implementation of the script method **onKeyboardCapture()** looks like this:

```
function onKeyboardCapture( captureName, data )
  app.send( "Keyboard Capture: " .. captureName .. "," .. data .. "{Enter}" )
end
```

Remember that you can have one or more USB-HID scanners connected to your PC as well as a serial device at the same time.

## Record Key Sequence

The **Record Key Sequence** dialog is used for recording key definitions, hot keys, and prefix and suffix of keyboard captures.




The **New** button removes any recorded key events and starts the recorder. The list of events is not updated during the recording.

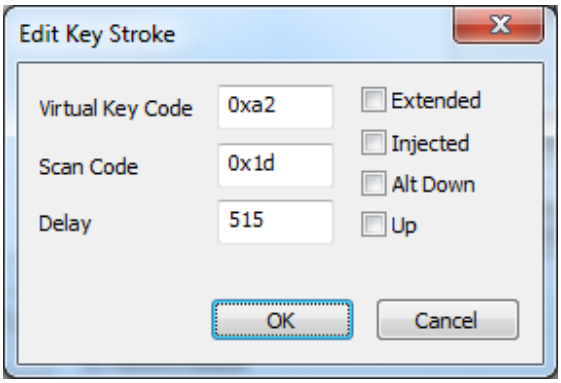
The **Add** button also starts the recorder, but new events are added to the list instead of starting a new key sequence.

The **Stop** button stops an ongoing recording. **Clear** removes all key events from the list. By using the **Play** button you can replay a recorded key sequence for test purposes. When replaying, first a pause of 3 seconds is made, giving you time to set focus to the receiving application. If you need more time, please put an additional delay for the first key event.

Remove the check from **Filter Keys During Recording** if you want the keys to be executed while recording.

Check **Record Delays** to record the delays between key presses. When replaying a key sequence that was recorded with this option on, it will be replayed at the same "typing" speed as when recorded.

Selecting a key event in the list and clicking on the **Edit...** button will let you modify the properties of the key event:




This is the information for the down event of the Ctrl key recorded.

### Virtual Key Code

Internally Windows has a table of values where each value corresponds to a key. This table is different from the code that the keyboard hardware sends to the PC. Applications in Windows use virtual key codes as input for text and for other keys.

### Scan Code

A scan code is the raw code that the keyboard hardware send to the PC. Most, but not all, applications don't care about this information.

### Delay

The delay in milliseconds that should performed before this key event is replayed.

### Extended

Some keys on a keyboard are called extended keys. For example, the extended flag is used to distinguish between the Enter key on the main part of the keyboard and the Enter key on the numeric keypad.

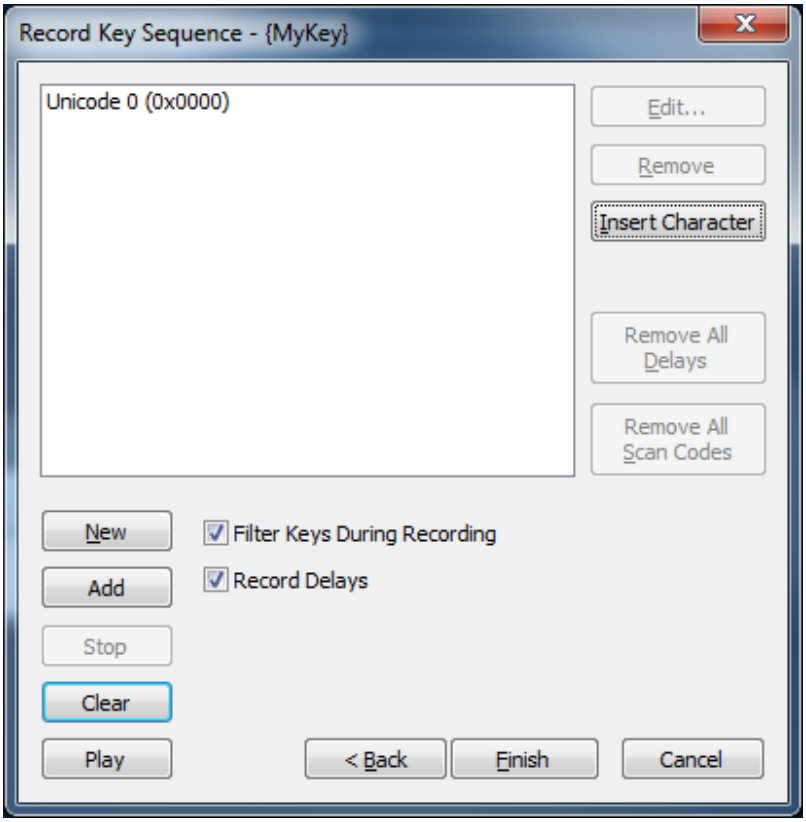
### Injected & Alt Down

You should never have to modify these flags.

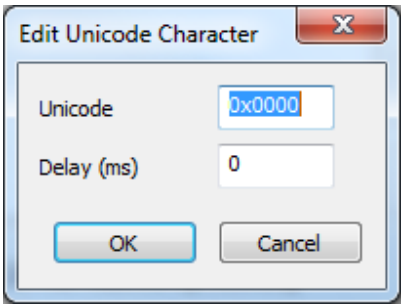
### Up

When checked, the key event is a key release event. Otherwise it is a key press event.

The key event list in the **Record Key Sequence** dialog can also contain character elements. To insert a character element, click on the **Insert Character** button.

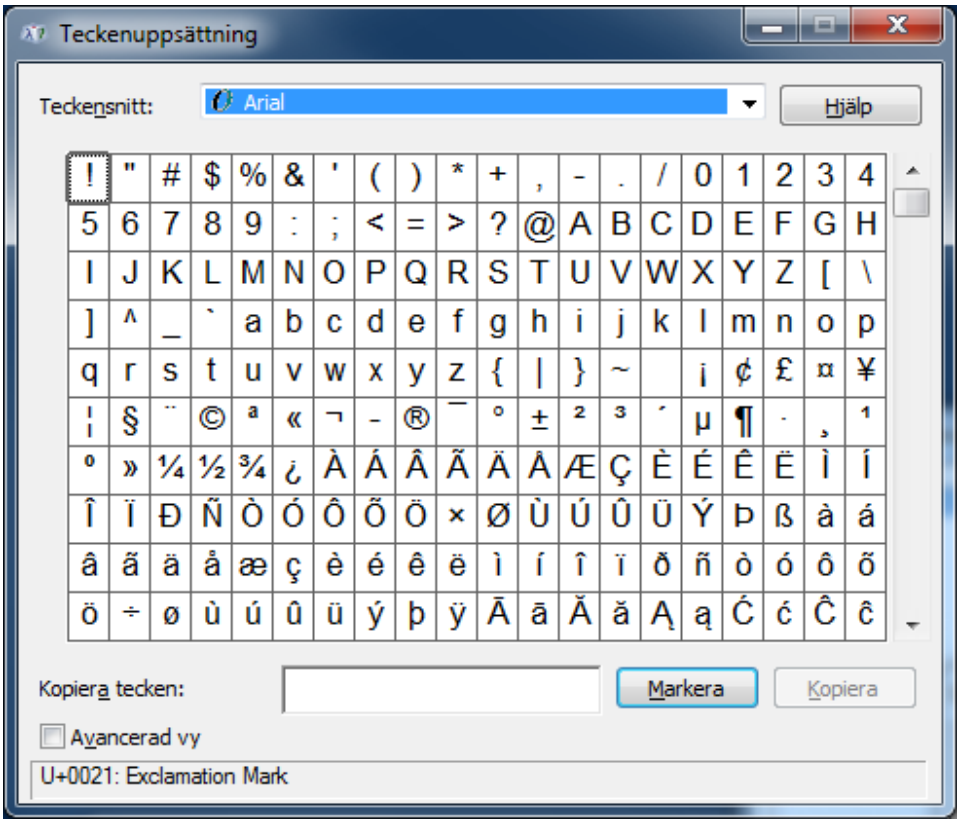



Then select the character element and click on the **Edit...** button.



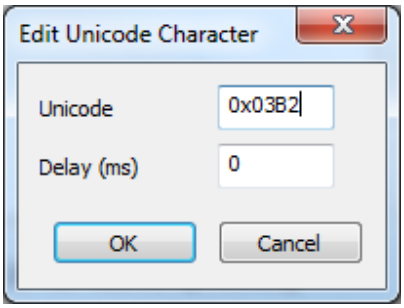

In the **Unicode** edit box, enter the Unicode character value for the character you want produced when the key sequence is replayed. The value should be a four digit hexadecimal value.

You can use the Windows utility program Character Map to find out the Unicode character value for international and special characters.



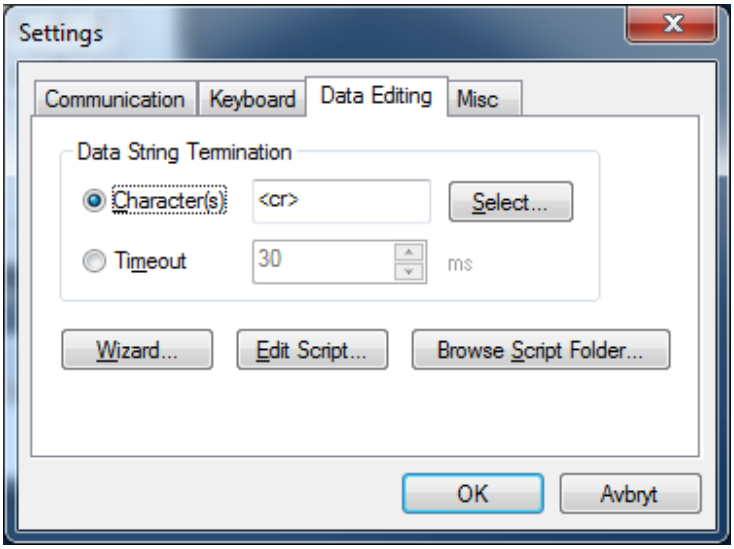

For example: to add a character element for the Greek letter β (beta), scroll down in the Character Map window until you find the character. Click on it and note that the character's Unicode value is displayed in the status bar of the Character Map window.

The value is 0x03B2. Enter this value in the **Unicode** edit box:



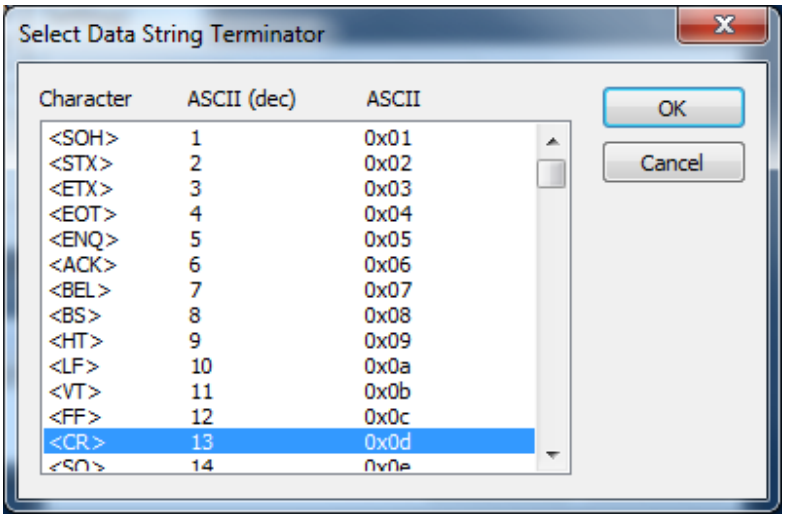

A key sequence can contain both recorded key events and character elements.

## Data Editing Tab

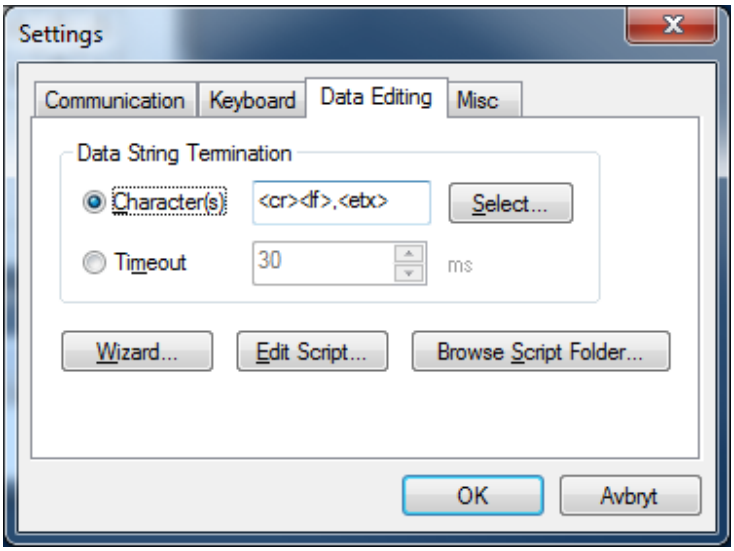



The **Data String Termination** setting tells Freefloat Link which character or character sequence terminates a data string received on the serial port. Alternatively, you can use a timeout value as a terminator. This means that if **Timeout** is set to 30 ms, Freefloat Link will terminate an input string when no data has been received for 30 ms.

Clicking on **Select** will display a list of characters from which you can choose the terminator:



A terminator can be made up of more than one character and you can have multiple terminators. For example: you have a device that terminates the data string with <cr><lf> sometimes and <etx> sometimes. Then the terminator would look like this:



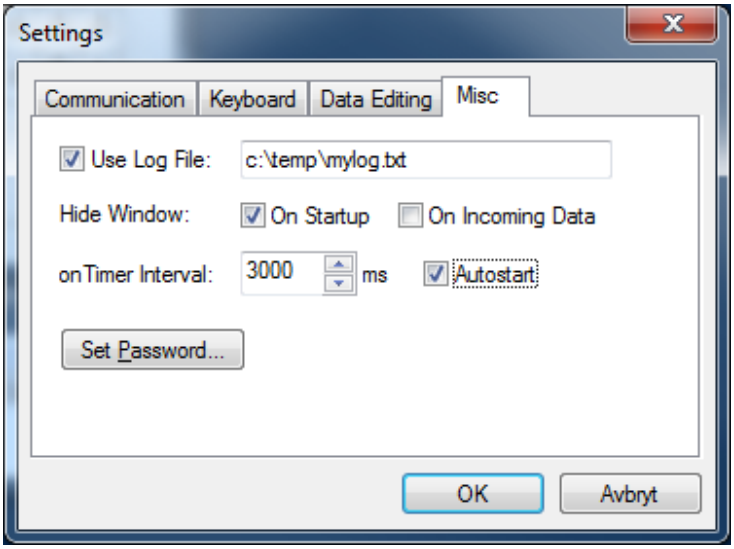

The **Edit Script...** button opens the Freefloat Link script. The name of the script file is Script.txt and it is opened in the associated program, normally Notepad. The text editor SciTE knows the syntax of Lua. It might be useful when writing Freefloat Link scripts.

The button **Browse Script Folder...** will open Windows Explorer in the folder that contains the script, configuration, and the license file.

The button **Wizard...** starts the wizard. Please see the topic **The Wizard** for information about the wizard.

For information about scripting in Freefloat Link see the topic **Freefloat Link Scripting**.

## Misc Tab




If you would like all the internal messages and events in Freefloat Link to be written to a log, check the setting **Use Log File** and enter a valid path and filename in the edit box. The log file is mostly used for troubleshooting a script.

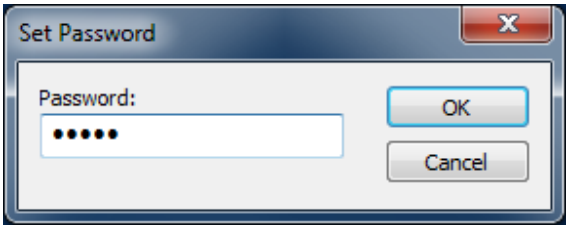
When you check **On Startup**, Freefloat Link's main window will be hidden on startup. When you check **On Incoming Data**, it will be hidden when serial data is received.

When a value greater than 0 is entered into **onTimer Interval**, the script method **onTimer()** will be called once during the specified time interval. For example: if you enter the value 3000, **onTimer()** will be called once every third second. Please take care when choosing a value here, if you enter 1 ms, **onTimer() will** be called 1000 times per second. This could make your PC unresponsive. Of course this will be highly dependant on what code the **onTimer()** method contains.

When the option **Autostart** is checked, Freefloat Link will be started automatically when you login to Windows. Freefloat Link will then be started with the profile for which you activated **Autostart**. If you have two profiles, one serving COM1 and the other COM2, you can check Autostart for each of those profiles. One instance of Freefloat Link will be started at login for every profile that has **Autostart** checked.

## Set Password Dialog

When you click **Set Password...** button on the **Misc** tab the dialog **Set Password** is displayed.




Here you can set a password that is required when exiting Freefloat Link and when clicking on the **Settings...** button in the main window. To remove the password, remove any text in the input box and click on **OK**.

## Where Are My Settings?

A Freefloat Link configuration consist of mainly two parts, the settings (serial port configuration, hot keys, etc.) and the script file(s).

As of version 5.1 the settings are no longer stored in the registry. They were made file based to enable different users on the same PC to share the same Freefloat Link configuration. The configuration is stored in the file Config.dat. Do not edit this file manually.

In version 5.1, the utility **Settings Exporter** was removed since it is no longer needed.

To determine where script and configuration files are located, click on the **Browse Script Folder...** button in the **Data Editing** tab of the **Settings** dialog.

## Freefloat Link  Scripting

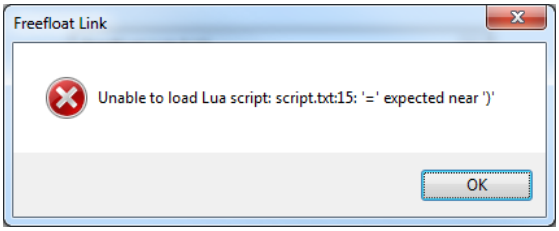
### Overview

Freefloat Link has an embedded script language called Lua. When Freefloat Link receives data from a device, a hot key is pressed etc. certain methods in the script are called. The code in these scripts determines what action is taken.

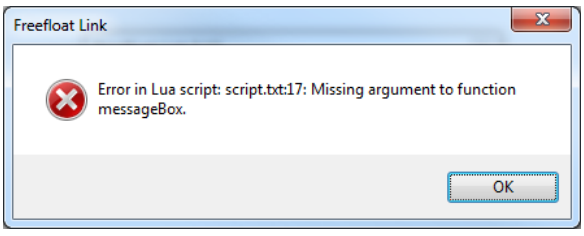
The name of the script file is Script.txt and is placed in the Freefloat Link application data folder. The location of this folder varies depending on what operating system you are using. If you need to make a backup of the script or copy it to another PC, click on the **Browse Script Folder...** on the **Misc** tab in the **Settings** dialog. Windows Explorer is opened and displays the contents of the script folder.

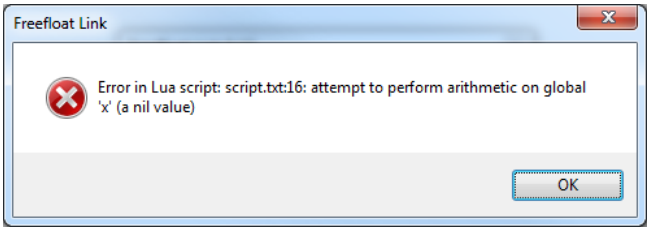
When you edit the script, remember to restart Freefloat Link to recompile the script or use the faster alternative of entering the **Settings** dialog and then exiting it.

If you make a mistake, for example create a syntax error, an error message is displayed when the script is compiled:




Also, some errors can appear when the script is running, so called run-time errors. Here are a couple of examples:



## The Lua Language

From http://www.lua.org/about.html:

*Lua is a powerful, fast, light-weight, embeddable scripting language.*

*Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics.*

Put simply, Lua is what makes data processing in Freefloat Link very flexible and powerful.

The reference manual for Lua can be found at the Lua site:

http://www.lua.org/

There is also a printed book on the Lua language, called **Programming in Lua**, which is more accessible than the reference manual.

Apart from Lua and its built-in language, Freefloat Link exposes a number of useful methods to the script.

## Script Events

When things happen in Freefloat Link  for example a hot key or a data string is received on the serial port, an event is generated. This results in a script method being called. The methods called when events happen are called *event methods*.

The table below is an overview and short description of all the different event methods. For a more detailed explanation, see the topic **Event Methods** below.

| Event Handler | When Called |
|---|---|
| onStart | Freefloat Link is started |
| onEnd | Freefloat Link is exited |
| onData | A data string is received on the serial port |
| onHotKey | A hot key is pressed |
| onKeyboardCapture | A data string is received from a HID device |
| onExternalData | A data string is received from an external application |
| onTimer | The timer interval has elapsed |
| onCTS | Status change on CTS |
| onDSR | Status change on DTR |
| onRI | Status change on RI |
| onDCD | Status change on DCD |

## Event Methods

In this topic all the event methods  are explained in detail.

### onStart()

This method is called when Freefloat Link is started. It is also called when you exit the **Settings** dialog.

This method receives no arguments.

#### Example: Beep on start

```
function onStart()
  -- Issue a short beep (3000 Hz, 50 ms)
  app.beep( 3000, 50 )
end
```

### onEnd()

Called when Freefloat Link is exited. It is also called when you enter the **Settings** dialog.

This method receives no arguments.

#### Example: Beep on exit

```
function onEnd()
  -- Issue a short beep (1000 Hz, 50 ms)
  app.beep( 1000, 50 )
end
```

### onData( data, length )

Called when a data string is received from the serial port.

This method receives the data string in *data* and the length of the string in *length*.

Data may contain binary characters including the null character.

Please note that if the **Data String Termination** is set to be a character and that character does not match the terminator used by your serial device, this method is never called.

#### Example: Hex dump of serial data

```
function onData( data, length )
  local numbers = ""
  local text = ""

  -- Loop for each character in data
  for i=1,length do
    -- Append character to text part
    if string.byte( data, i ) >= 32 then
      text = text .. string.sub( data, i, i )
    else
      -- Control characters are replace with '.'
      text = text .. "."
    end

    -- Add hex representation of the character
    numbers = numbers .. string.format( "%02x ", string.byte( data, i ) )

    -- Break lines at eight characters
    if (i % 8) == 0 then
      app.send( numbers .. text .. "{Enter}" )
      numbers = ""
      text = ""
    end
  end

  -- Handle the tail of the dump
  local c = length
  if (c % 8) ~= 0 then
    while (c % 8) ~= 0 do
      numbers = numbers .. "   "
      c = c + 1
    end
    app.send( numbers .. text .. "{Enter}" )
  end
end
```

To test the above example, copy and paste the code into the script replacing the default implementation of **onData()**. Use **Timeout** as the **Data String Terminator**. Connect a serial device to the serial port and make it generate some data. Below is the output when reading a barcode containing "W1711010814107013621" using a serial barcode reader:

```
57 31 37 31 31 30 31 30 W1711010
38 31 34 31 30 37 30 31 81410701
33 36 32 31 0d           3621.
```

The last character `0d` (hexadecimal) is the same as 13 in decimal notation. The ASCII character with code 13 is carriage return. This means the barcode reader is using carriage return (<cr>) as its data string terminator.

## onHotKey( name )

This method is called when you press a hot key.

The argument to this method is the name of the hot key that was pressed.

### Example: Message box displaying the hot key's name

```
function onHotKey( name )
  app.messageBox( "Hot Key Pressed:", name )
end
```

When executed, the above method will display a message box with the name of the hot key:

### onKeyboardCapture( name, data )

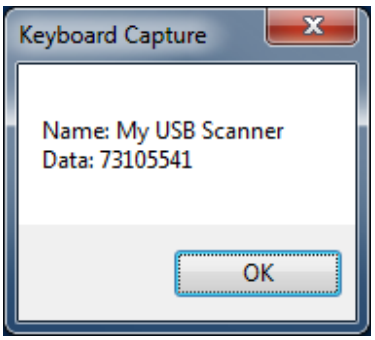
Called when a keyboard capture string has been received.

The arguments to this method are the name of the keyboard capture and the data.

#### Example: Display the name and data of a keyboard capture event

```
function onKeyboardCapture( name, data )
  app.messageBox( "Keyboard Capture", "Name: " .. name .. "\n" ..
    "Data: " .. data )
end
```

If you have a keyboard captured defined called *My USB Scanner* and it captures the string *73105541* the method in the above example will display this dialog:




### onExternalData( data, length )

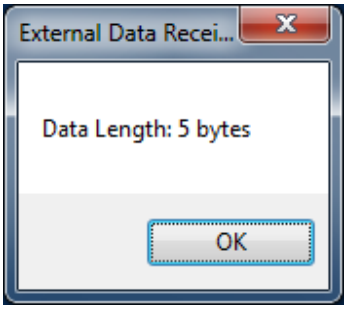
Called when an external application sends data to Freefloat Link.

The arguments are the received data and the length of it.

External applications can send data to Freefloat Link. They do it by finding the window handle of Freefloat Link's window and then send a WM_COPYDATA message to the window.

#### Example: Display data and length sent to Freefloat Link from an external application

```
function onExternalData( data, length )
  app.messageBox( "External Data Received", "Data Length: " .. length .. " bytes" )
end
```

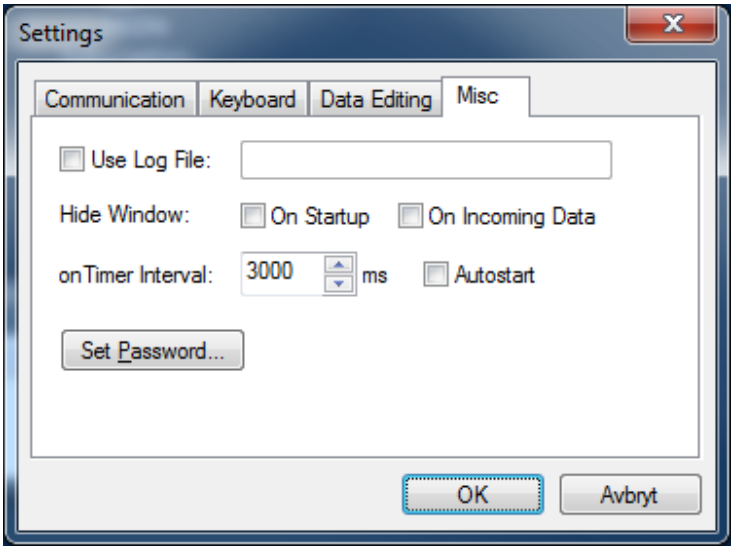


This feature makes it possible to create an application that integrates tightly with Freefloat Link. For example, the code in **onExternalData()** could relay the data to a scanner to make it beep, initiate a scan, configure it etc.

## onTimer()

Called periodically at the specified timer interval.

This method receives no arguments.




In the example above, the **onTimer Interval** has been set to 3000 milliseconds (3 seconds). This means that the **onTimer()** script method will be called once every three seconds.

This method can be used for adding timeout logic to a solution.

### Example: Output a time stamp to the active application at the onTimer interval

```
function onTimer()
  app.send( os.date( "%H:%M:%S" ) .. "{Enter}" )
end
```

### onCTS( status )

This method is called when the hardware handshake signal changes state.

The argument *status* is **true** when the signal goes high and **false** when the signal goes low.

This description also applies to onDSR, onRI, and onDCD.

#### Example: Outputs the state of CTS when it is changed

```
function convertSignal( status )
  if status then
    return "High"
  else
    return "Low"
  end
end

function onCTS( status )
  app.send( "CTS " .. convertSignal( status ) .. "{Enter}" )
end
```

## Script Methods

Lua is a generic script language and has methods to manipulate string, tables, files, and so on. However, it does not contain any methods to retrieve the title of a window, simulate keys etc. So in order to activate an application window, send key strokes and similar operations, a number of internal methods in Freefloat Link have been exposed to the embedded Lua script engine.

The tables below offers an overview of the methods. The methods have been grouped into areas of interest for easier reference. Below the tables there is a reference section with a detailed description of each method.

When these methods are used in a script, you need to prefix them with "app.", for example `app.sleep( 100 )`.

| **Output/User Feedback** | |
|---|---|
| **Method** | **Description** |
| beep | Beeps with the internal PC speaker |
| blinkIcon | Changes the color of the notification area icon |
| log | Writes a text line to a log file |
| messageBox | Displays a message box |
| playSound | Plays a sound file |
| playSystemSound | Plays a sound associated with a system event |
| send | Sends keyboard data |
| sendSerialData | Sends serial data to the COM-port |
| sendSubscriberData | Sends data to subscribers |

| **Windows** | |
|---|---|
| **Method** | **Description** |
| enumWindows | Enumerates all windows |
| findWindow | Finds a window with the specified title and/or class |
| getForegroundWindow | Gets the handle of the foreground window |
| getWindowClass | Gets the class name of a window |
| getWindowText | Gets the title of a window |
| setForegroundWindow | Sets a window to be the foreground window |
| setWindowText | Sets the title of a window |
| windowOperation | Executes a window operation |

| **Clipboard** | |
|---|---|
| **Method** | **Description** |
| getClipboardData | Gets the text on the clipboard |
| setClipboardData | Sets the text on the clipboard |

| Application Launch | |
|---|---|
| **Method** | **Description** |
| closeAppHandle | Closes the application handle |
| isAppRunning | Determines if an application is still running |
| run | Starts a program |

| Serial Port | |
|---|---|
| **Method** | **Description** |
| closePort | Closes the serial port |
| getDTR | Gets the status of the DTR signal |
| getRTS | Gets the status of the RTS signal |
| openPort | Opens the serial port |
| setDTR | Sets the status of the DTR signal |
| setRTS | Sets the status of the RTS signal |

| Miscellaneous | |
|---|---|
| **Method** | **Description** |
| ean128 | Parses the contents of a GS1-128/UCC/EAN-128 code |
| gs1128 | |
| exit | Exits Freefloat Link |
| exitWindows | Either logs off the current user, shuts down the PC, or shuts down and restarts the PC |
| getProfile | Gets the current Freefloat Link profile |
| getTickCount | Gets the number of milliseconds elapsed since the system was started |
| lockWorkStation | Locks Windows |
| setProfile | Sets the current profile |
| setTimer | Sets the script timer |
| sleep | Delays the script for some time |

## Output/User Feedback

### *beep( frequency, duration )*

#### Description

Makes the internal PC speaker beep with the specified *frequency* and *duration*.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *frequency* | Number | The frequency of the beep in Hertz. |
| *duration* | Number | The duration of the beep in milliseconds. |

#### Returns

true if successful, false otherwise.

#### Constants

None.

#### Example

```
-- Issue a short beep (3000 Hz, 50 ms) when Freefloat Link is started
function onStart()
  app.beep( 3000, 50 )
end
```

### *blinkIcon( icon, duration )*

#### Description

Changes the notification *icon* color for the specified *duration*. After the duration has elapsed the icon will return to the default color grey.

The operation is asynchronous, in other words, if blinkIcon is called again before the duration for the first call has elapsed, the new icon is set immediately by the second call.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *icon* | Number | The icon color index. |
| *duration* | Number | The duration in milliseconds the color should be displayed before restoring the default color. |

#### Returns

Nothing.

#### Constants

```
iconColors =
{
  ["GREY"] = 0,
  ["GREEN"] = 1,
  ["YELLOW"] = 2,
  ["BLUE"] = 3,
  ["RED"] = 4,
}
```

#### Example

Please note the delay in the loop is needed to display each icon color for half a second.

```
-- Cycle through icon colors
function onStart()
  for i = iconColors["GREY"], iconColors["RED"] do
    app.blinkIcon( i, 500 )
    app.sleep( 500 )
  end
end
```

### *log( filename, message )*

#### Description

Writes the message to the specified log file. Note that logging with this method from the script is separate from the built-in logging facility.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *filename* | String | The path and name of the log file to write to. |
| *message* | String | The log message to write. |

#### Returns

Nothing.

#### Example

```
function onStart()
  app.log( "c:\\myscript.log", "onStart() called" )
end
```

### *messageBox( title, message, type )*

#### Description

Displays a message box with the specified *title* and *message*. The *type* argument specifies the number and type of buttons used.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *title* | String | The message box title. |
| *message* | String | The message to be displayed. |
| *type* | Number, optional | The type of buttons to use. |

#### Returns

A value from the constants in mbResults (see below) indicating the button clicked.

#### Constants

Use one of the following constants for the *type* argument:

```
mbTypes =
{
  ["OK"] = 0,
  ["OKCANCEL"] = 1,
  ["YESNOCANCEL"] = 3,
  ["YESNO"] = 4,
  ["ICONHAND"] = 16,
  ["ICONQUESTION"] = 32,
  ["ICONEXCLAMATION"] = 48,
  ["ICONASTERISK"] = 64,
  ["DEFBUTTON1"] = 0,
  ["DEFBUTTON2"] = 256,
  ["DEFBUTTON3"] = 512,
}
```

The possible return values are:

```
mbResults =
{
  ["OK"] = 1,
  ["CANCEL"] = 2,
  ["YES"] = 6,
  ["NO"] = 7,
}
```

Example

```
-- Ask the user if she/he wants to exit Freefloat Link and acts on the answer
function onHotKey( name )
  local result = app.messageBox( "Freefloat Link Script",
    "Are you sure you want to exit?",
    mbTypes["YESNO"] + mbTypes["DEFBUTTON2"] )

  if mbResults["YES"] == result then
    app.exit()
  end
end
```

### *playSound( filename, options )*

#### Description

Plays the sound specified by the *filename* argument.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *filename* | String | The path and filename of the sound file. |
| *options* | Number | Specifies the behavior for the sound playback. Use the constants in *soundOptions*. |

#### Returns

true if successful, false otherwise.

#### Constants

```
soundOptions =
{
  ["SYNC"] = 0,
  ["ASYNC"] = 1,
  ["NODEFAULT"] = 2,
  ["LOOP"] = 8,
  ["NOSTOP"] = 16,
  ["NOWAIT"] = 8192,
}
```

#### Example

```
-- Buzz like a bee
function onHotKey( name )
  app.playSound(
    "c:\\windows\\system32\\buzzingbee.wav", soundOptions["SYNC"] )
end
```

### *playSystemSound( systemEvent, options )*

#### Description

Plays the sound specified by the given system event. Different system events can be mapped to sound files in the Control Panel.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *systemEvent* | Number | The ID of the system event. |
| *options* | Number | Specifies the behavior for the sound playback. Use the constants in *soundOptions*. |

#### Returns

true if successful, false otherwise.

#### Constants

Use one of the following values for the *systemEvent* argument:

```
systemSounds =
{
  ["ASTERISK"] = 10835,
  ["QUESTION"] = 16211,
  ["HAND"] = 18515,
  ["EXIT"] = 17747,
  ["START"] = 21331,
  ["WELCOME"] = 22355,
  ["EXCLAMATION"] = 8531,
  ["DEFAULT"] = 17491,
}
```

The possible values of the *options* argument:

```
soundOptions =
{
  ["SYNC"] = 0,
  ["ASYNC"] = 1,
  ["NODEFAULT"] = 2,
  ["LOOP"] = 8,
  ["NOSTOP"] = 16,
  ["NOWAIT"] = 8192,
}
```

#### Example

```
-- Play the sound mapped to the system event Exclamation
function onHotKey( name )
  app.playSystemSound( systemSounds["EXCLAMATION"], soundOptions["SYNC"] )
end
```

### *send( data )*

#### Description

Sends keyboard data.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *data* | String | The data to send. |

The argument *data* is a string consisting of text, key names, and Unicode characters.

| Type | Sample String | Description |
|---|---|---|
| Text | "Rob was here" | Regular characters. |
| Key Name | "{Enter}" | Key names corresponds to the keys defined in the dialog **Key Settings**. |
| Unicode | "{65}" | Character in Unicode decimal notation. |
| Unicode | "{0x0041}" | Character in Unicode hexadecimal notation. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
function onHotKey( name )
  -- Send a regular string
  app.send( "Rob was here!" )

  -- Send the characters ABC by using Unicode notation
   app.send( "{0x0041}{0x0042}{0x0043}" )

  -- Send enter using its key name
  app.send( "{Enter}" )
end
```

**Note!**

- When you want **send()** to send the characters \ and { you need to escape those with a backslash. There is a helper method called **escapeData()** in the supplied file extra.lua.
- If you specify a key name for a key that is not defined, it will be ignored.
- When sending data to certain applications they might miss key presses if the key events are sent too fast or too early. You may need to increase the setting **Interkey Delay** and/or intersperse calls to **send()** with calls to **sleep()** or **findWindow()** depending on the situation.

### *sendSerialData( data, length )*

#### Description

Sends serial data to the COM-port.

Note that Freefloat Link needs to be configured to use a COM-port for this method to work.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *data* | String | The data to send to the COM-port. |
| *length* | Number | The number of characters of *data* that should be sent. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
-- Send a binary string to the COM-port
function onHotKey( name )
  -- Build a string containing the characters null, soh, and stx
  local s = string.char( 0 ) .. string.char( 1 ) .. string.char( 2 )
  app.sendSerialData( s, 3 )
end
```

### *sendSubscriberData( data, length )*

#### Description

Sends data to subscribers.

If there are no subscribers, calling this method has no effect.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *data* | String | The data to send to subscribers. |
| *length* | Number | The number of characters of *data* that should be sent. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
function onHotKey( name )
  -- Send a string to all connected subscribers
  local s = "Hello Subscriber!"

  app.sendSubscriberData( s, string.len( s ) )
end
```

## Windows

### *enumWindows( handle )*

#### Description

Enumerates all windows.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle to the window whose child windows should be enumerated. Specify null (0) to enumerate all top level windows. |

#### Returns

A table containing all the window handles of the enumerated windows.

#### Constants

None.

#### Example

```
-- Enumerate all top level windows, place handle values and window titles on clipboard
function onHotKey( name )
  local t = app.enumWindows( 0 )
  local stot = ""

  for k, v in pairs( t ) do
    s = string.format( "%08x", v )
    stot = stot .. s .. ": " .. app.getWindowText( v ) .. "\r\n"
  end

  -- Paste the clipboard into a program to display the result
  app.setClipboardData( stot )
end
```

### *findWindow( title, class )*

#### Description

Finds a window with the specified title and class.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *title* | String | The title of the sought window. |
| *class* | String, optional | The window class of the sought window. |

#### Returns

The window handle if the window is found or null otherwise.

#### Constants

None.

#### Example

```
function onHotKey( name )
  -- Check if Notepad is running (with no doc name given)
  if app.findWindow( "Untitled - Notepad" ) then
    app.messageBox( "Freefloat Link Script", "Notepad is running" )
  else
    app.messageBox( "Freefloat Link Script", "Notepad is not running" )
  end
end
```

### *getForegroundWindow()*

#### Description

Gets the handle of the foreground window.

#### Arguments

None.

#### Returns

The window handle of the foreground window. In special circumstances this can be null so you need to check the return value before further use of it.

#### Constants

None.

#### Example

```
function onHotKey( name )
  local handle = app.getForegroundWindow()

  if handle ~= 0 then
    app.messageBox( "Freefloat Link Script", app.getWindowText( handle )  )
  end
end
```

### *getWindowClass( handle )*

#### Description

Gets the class name of the specified window.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle of the window. |

#### Returns

A string containing the class name of the window or an empty string if the class name couldn't be retrieved.

#### Constants

None.

#### Example

```
-- Displays the class name of the foreground window
function onHotKey( name )
  local class = app.getWindowClass( app.getForegroundWindow() )
  app.messageBox( "Freefloat Link Script", class )
end
```

### *getWindowText( handle )*

#### Description

Gets the title of the specified window. This method also works on child windows such as buttons, edit boxes, and similar controls.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle of the window. |

#### Returns

A string containing the window title of the window or an empty string if the window text couldn't be retrieved.

#### Constants

None.

#### Example

```
-- Displays the window title of the foreground window
function onHotKey( name )
  local title = app.getWindowText( app.getForegroundWindow() )
  app.messageBox( "Freefloat Link Script", title )
end
```

### *setForegroundWindow( handle )*

#### Description

Sets the specified window to be the foreground window.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle of the window. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
-- Bring the Windows Media Player window to the foreground
function onHotKey( name )
 local handle = app.findWindow( "Windows Media Player" )
 if handle ~= 0 then
   app.setForegroundWindow( handle )
 end
end
```

### *setWindowText( handle, text )*

#### Description

Sets the title of the specified window. This method also works on child windows such as buttons, edit boxes, and similar controls.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle of the window. |
| *text* | String | The title to set. |

#### Returns

true if successful, false otherwise.

#### Constants

None.

#### Example

```
-- Set a new title for a Notepad window
function onHotKey( name )
 local handle = app.findWindow( "Untitled - Notepad" )
 if handle ~= 0 then
   app.setWindowText( handle, "My Text Editor" )
 end
end
```

### *windowOperation( handle, operation )*

#### Description

Executes a window operation.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The handle of the window. |
| *operation* | Number | The operation to perform on the window. |

#### Returns

Nothing.

#### Constants

The following constants define the possible values of the *operation* argument:

```
windowOperations =
{
 ["CLOSE"] = 1,
 ["MAXIMIZE"] = 4,
 ["MINIMIZE"] = 5,
 ["RESTORE"] = 6,
 ["TASKLIST"] = 9,
 ["MONITORPOWER"] = 11,
 ["SCREENSAVE"] = 12,
}
```

#### Example

```
-- Maximizes a Notepad window then restores it
function onHotKey( name )
 local handle = app.findWindow( "Untitled - Notepad" )
 if handle ~= 0 then
   app.windowOperation( handle, windowOperations["MAXIMIZE"] )
   app.sleep( 2000 )
   app.windowOperation( handle, windowOperations["RESTORE"] )
 end
end
```

## Clipboard

### *getClipboardData()*

#### Description

Gets the text from the clipboard.

#### Arguments

None.

#### Returns

The text contents on the clipboard as a string and the length of the data. If the call fails or the clipboard doesn't have any text, an empty string and zero length is returned. Note that the terminating null is counted.

#### Constants

None.

#### Example

```
-- Displays the text length and content on the clipboard in a message box
function onHotKey( name )
  local text, textlen = app.getClipboardData()
  app.messageBox( "Clipboard Contents",
    string.format( "%d characters\r\n", textlen ) .. text )
end
```

### *setClipboardData( text )*

#### Description

Sets the text on the clipboard.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *text* | String | The text to set. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
-- Sets the text contents of the clipboard and then retrieves it
function onHotKey( name )
 local s = "ABC"
 app.setClipboardData( s )

 local text, textlen = app.getClipboardData()
 app.messageBox( "Clipboard Contents", string.format( "%d characters\r\n", textlen
) .. text )
end
```

## Application Launch

### *closeAppHandle( handle )*

#### Description

Closes the application handle.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The application handle. |

#### Returns

Nothing.

#### Constants

None.

#### Example

See the **run()** method.

### *isAppRunning( handle )*

#### Description

Determines if an application is still running.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *handle* | Number | The application handle. |

#### Returns

Nothing.

#### Constants

true if the application is still running, false otherwise.

#### Example

See the **run()** method.

### *run( program, argument, delay )*

#### Description

Starts a program with the specified arguments.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *program* | String | The full path to the executable. |
| *argument* | String | The command line argument string. |
| *delay* | Number, optional | The number of milliseconds to wait until the started application is waiting for user input. If this argument isn't specified, the default wait time is 10000 ms. |

#### Returns

The application handle. Note that this handle needs to be closed with the method **closeAppHandle()** to avoid memory leaks.

If the application couldn't be started, a runtime error occurs.

#### Constants

None.

#### Example

```
function onHotKey( name )
  local appHandle = app.run( "c:\\windows\\notepad.exe", "c:\\test.txt" )

  -- As an extra precaution
  app.sleep( 500 )

  -- Send some text to Notepad
  app.send( "Hi!" )

  -- Wait until the user exits Notepad
  while app.isAppRunning( appHandle ) do
    app.sleep( 100 )
  end

  -- Close the handle
  app.closeAppHandle( appHandle )

  -- Confirm exit with a message
  app.messageBox( "Freefloat Link Script", "Notepad is dead" )
end
```

## Serial Port

### *closePort()*

#### Description

Closes the serial port.

#### Arguments

None.

#### Returns

Nothing.

#### Constants

None.

#### Example

See the **openPort()** method.

### *getDTR()*

#### Description

Gets the status of the DTR signal of the serial port. DTR is an output signal.

#### Arguments

None.

#### Returns

A boolean which indicates the DTR signal status (true = high, false = low).

#### Constants

None.

#### Example

```
-- Display the status of the DTR signal
function displayDTRStatus()
  local s = "off"

  if app.getDTR() then
    s = "on"
  end

  app.messageBox( "Freefloat Link Script", "DTR is " .. s )
end

function onHotKey( name )
  -- Display default (from settings)
  displayDTRStatus()

  -- Modify status and display it
  app.setDTR( false )
  displayDTRStatus()
  app.setDTR( true )
  displayDTRStatus()
end
```

### *getRTS()*

#### Description

Gets the status of the RTS signal of the serial port. RTS is an output signal.

#### Arguments

None.

#### Returns

A boolean which indicates the RTS signal status (true = high, false = low).

#### Constants

None.

#### Example

See the **getDTR()** method.

### *openPort()*

#### Description

Opens the serial port.

**openPort()** and **closePort()** can be used when you need to release the serial port, start an application that uses the port for a while, and then reopen the port.

#### Arguments

None.

#### Returns

A boolean indicating if the port could be opened. If the port is being held open by another application a call to this method will return false .

#### Constants

None.

#### Example

**Note!** This is only a code fragment.

```
function onHotKey( name )
  -- Close the serial port to let the external application use it
  app.closePort()

  Start external application and wait for it to exit

  -- Reopen the serial port
  app.openPort()
end
```

### *setDTR( status )*

#### Description

Sets the status of the DTR signal. DTR is an output signal.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *status* | Boolean | The status to set, (true = high, false = low). |

#### Returns

Nothing.

#### Constants

None.

#### Example

See the **getDTR()** method.

### *setRTS( status )*

#### Description

Sets the status of the RTS signal. RTS is an output signal.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *status* | Boolean | The status to set, (true = high, false = low). |

#### Returns

Nothing.

#### Constants

None.

#### Example

See the **getDTR()** method.

## Miscellaneous

### *ean128( data, strict )*

#### Description

Parses the contents of a GS1-128 code (earlier called UCC-128 or EAN-128). For variable length fields followed by another field, the data must be delimited by a Group Separator (GS, ASCII 29, hex 1D).

An alternative name of this method is **gs1128()**.

Please refer to http://www.gs1.org for information about GS1 Application Identifiers.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *data* | String | The GS1-128 data to be parsed and split into separate fields. |
| *strict* | Boolean | In strict mode, spaces are not allowed in alphanumeric fields. |

#### Returns

A table where the keys are the Application Identifiers (AIs) and the values are the contents of the fields. If the parsing fails, a nil value is returned. The parsing can fail if the code is not a GS1 code or if the code doesn't follow the standard.

#### Constants

None.

#### Example

```
-- Parses and outputs a list of AIs and values
function onData( data, length )
  -- Parse the GS1 code
  fields = app.ean128( data, true )

  if fields then
    -- Output AIs and values
    for k,v in pairs(fields) do
      app.send( "AI: " .. k .. "  Value: " .. v .. "{Enter}" )
    end
    app.send( "{Enter}" )
  else
    app.messageBox( "Freefloat Link Script", "GS1 parsing failed." )
  end
end
```

### *exit()*

#### Description

Exits Freefloat Link. Please note that the exit is not immediate, Freefloat Link will exit as soon as the current script has finished executing.

#### Arguments

None.

#### Returns

Nothing.

#### Constants

None.

#### Example

```
function onHotKey( name )
  -- Will exit Freefloat Link as soon as this method has finished executing
  app.exit()

  app.messageBox( "Freefloat Link Script", "Goodbye!" )
end
```

### *exitWindows( options )*

#### Description

Either logs off the current user, shuts down the PC, or shuts down and restarts the PC.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *options* | Number, optional | Type of action to be performed. |

#### Returns

Nothing.

#### Constants

The following constants define the type of action to be performed:

```
exitWindowsOpts =
{
  ["LOGOFF"] = 0,
  ["SHUTDOWN"] = 1,
  ["REBOOT"] = 2,
  ["FORCE"] = 4,
  ["POWEROFF"] = 8,
}
```

Please note that the default value for *options* is Logoff (0). The Force (4) value must be used in combination with Logoff (0), Shutdown (1), Reboot (2), or Poweroff (8). Use Force (4) with care since it will end applications without asking the user to save data.

#### Example

```
-- Log off user
function onHotKey( name )
 app.exitWindows( exitWindowsOpts["LOGOFF"] )
end
```

### *getProfile()*

#### Description

Gets the current Freefloat Link profile.

#### Arguments

None.

#### Returns

A string containing the name of the current profile.

#### Constants

None.

#### Example

```
-- Display the current profile's name
function onHotKey( name )
  app.messageBox( "Freefloat Link Script", "Current Profile: " .. app.getProfile() )
end
```

### *getTickCount()*

#### Description

Gets the number of milliseconds elapsed since the system was started. This method can for example be used to take time between events in Freefloat Link.

#### Arguments

None.

#### Returns

The number of milliseconds elapsed since the system was started.

#### Constants

None.

#### Example

```
lastTime = 0

-- Displays the time between each hot key event
function onHotKey( name )
  -- Is this the first time the event happens?
  if lastTime == 0 then
    app.send( "First event.{Enter}" )
  else
    -- Display the time elapsed since last time this event happened
    local timeElapsed = app.getTickCount() - lastTime
    app.send( timeElapsed .. " ms{Enter}" )
  end

  -- Remember time stamp for future calls
  lastTime = app.getTickCount()
end
```

### *lockWorkStation()*

#### Description

Locks Windows.

#### Arguments

None.

#### Returns

true if successful, false otherwise.

#### Constants

None.

#### Example

```
-- Locks the Windows session
function onHotKey( name )
  app.lockWorkStation()
end
```

### *setProfile( profile )*

#### Description

Sets the current profile in Freefloat Link.

Please note that a profile change reinitializes the Lua script engine and because of this, any information held in global variables will be lost. If you need any information to survive over a profile switch, you will need to store it in a file.

The actual switch is delayed until the script has finished executing the current method.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *profile* | String | The name of the profile to switch to. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
myVar = 0

function onStart()
  app.messageBox( "onStart()", "myVar is " .. myVar )
  myVar = myVar + 1
end

-- Switches to the profile "My Profile"
function onHotKey( name )
  app.setProfile( "My Profile" )
end
```

### *setTimer( interval )*

#### Description

Sets the script timer to the specified interval. This is the same setting as on the **Misc** tab in the **Settings** dialog. To turn off the timer, specify zero as the interval.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *interval* | Number | The timer interval to set. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
-- When a hot key is pressed, sets the script timer interval
function onHotKey( name )
 app.setTimer( 1000 )
end
```

### *sleep( duration )*

#### Description

Delays the script for the specified time.

#### Arguments

| Argument | Type | Description |
|---|---|---|
| *duration* | Number | The time to wait. |

#### Returns

Nothing.

#### Constants

None.

#### Example

```
-- Outputs two periods with a one second pause between them
function onHotKey( name )
 app.send( "." )
 app.sleep( 1000 )
 app.send( "." )
end
```

## Notification Area Icon

When started, Freefloat Link adds an icon to the notification area (also called the Systray sometimes). It is used to indicate different states and events. Please note that the icon can also be modified by a script.

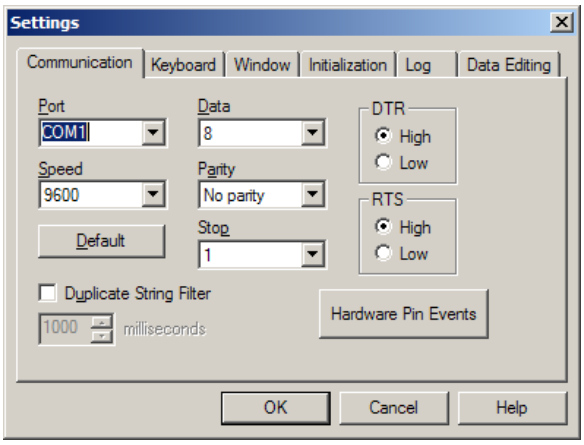
| Appearance | Explanation |
|---|---|
| | Freefloat Link is idle. |
| | Data was received from the serial port. |
| | Data was sent to the serial port. |
| | A serial hardware pin event was triggered.<br>**OR**<br>Data was received through a keyboard capture definition. |
| | The serial port specified in the profile could not be opened. |

## Migration guide WLinq 3.x to Freefloat Link

Freefloat Link is based on the earlier product called WLinq. Many functions present in WLinq (3.x) has been removed in Freefloat Link. The reason for this is to simplify Freefloat Link and to avoid confusion if there are more than one way to achieve a task.

This guide is meant to ease the transition from WLinq 3.x data formats to the new script based approach in Freefloat Link. Other types of features that has been affected are also explained in this chapter.

## Duplicate String Filter

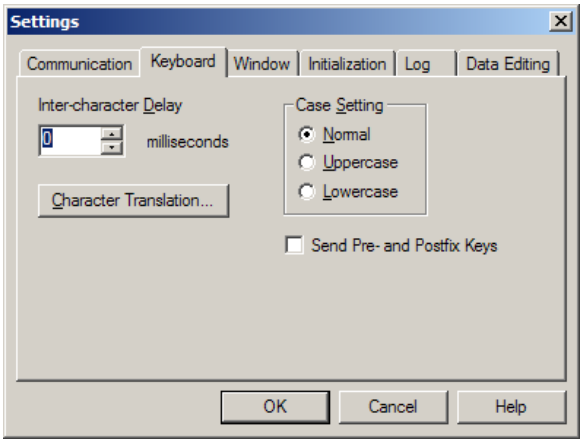



**WLinq 3.19 – Communication Property Page**

The **Duplicate String Filter** function has been removed. The equivalent function can be achieved in a script:

```
duplicateFilterTime = 1000
timeStamp = app.getTickCount()
lastCode = ""

function duplicateFilter( data )
  -- Calculate the time elapsed since last code was read
  elapsed = app.getTickCount() - timeStamp

  -- Do not filter the code if:
  --  the code is different OR
  --  the time elapsed since last code was read is greater than the duplicate filter time OR
  --  the timer has wrapped around (extremely rare)
  if (lastCode ~= data) or (elapsed > duplicateFilterTime) or (elapsed < 0) then
    -- Update last code and time
    lastCode = data
    timeStamp = app.getTickCount()

    -- Do not filter the code
    return false
  else
    -- Filter the code
    return true
  end
end

function onData( data, length )
  if not duplicateFilter( data ) then
    app.send( data .. "{Enter}" )
  end
end
```

## Case Setting




**WLinq 3.19 – Keyboard Property Page**

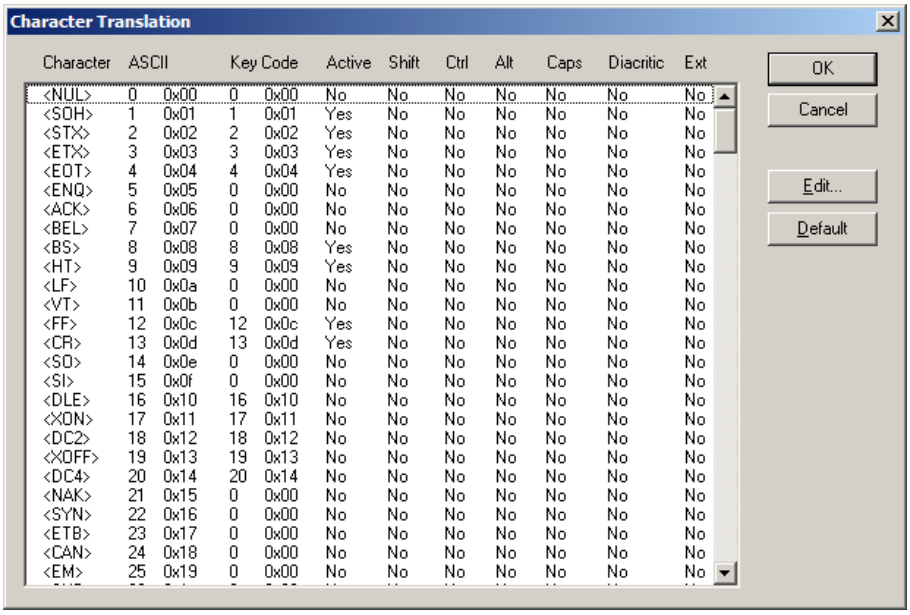
The **Case Setting** function has been removed. To achieve the same in a script, use the appropriate sample from below:

```
-- Normal Case
function onData( data, length )
  app.send( data .. "{Enter}" )
end
```

```
-- Upper Case
function onData( data, length )
  app.send( string.upper( data ) .. "{Enter}" )
end
```

```
-- Lower Case
function onData( data, length )
  app.send( string.lower( data ) .. "{Enter}" )
end
```

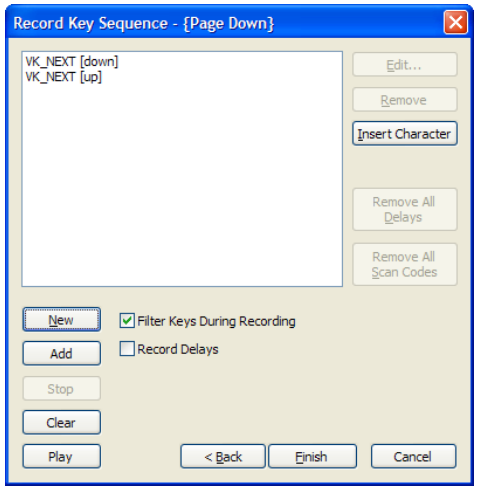
## Character Translation


Character Translation
Character ASCII Key Code Active Shift Ctrl Alt Caps Diacritic Ext
<NUL> 0 0x00 0 0x00 No No No No No No No
<SOH> 1 0x01 1 0x01 Yes No No No No No No
<STX> 2 0x02 2 0x02 Yes No No No No No No
<ETX> 3 0x03 3 0x03 Yes No No No No No No
<EOT> 4 0x04 4 0x04 Yes No No No No No No
<ENQ> 5 0x05 0 0x00 No No No No No No No
<ACK> 6 0x06 0 0x00 No No No No No No No
<BEL> 7 0x07 0 0x00 No No No No No No No
<BS> 8 0x08 8 0x08 Yes No No No No No No
<HT> 9 0x09 9 0x09 Yes No No No No No No
<LF> 10 0x0a 0 0x00 No No No No No No No
<VT> 11 0x0b 0 0x00 No No No No No No No
<FF> 12 0x0c 12 0x0c Yes No No No No No No
<CR> 13 0x0d 13 0x0d Yes No No No No No No
<SO> 14 0x0e 0 0x00 No No No No No No No
<SI> 15 0x0f 0 0x00 No No No No No No No
<DLE> 16 0x10 16 0x10 No No No No No No No
<XON> 17 0x11 17 0x11 No No No No No No No
<DC2> 18 0x12 18 0x12 No No No No No No No
<XOFF> 19 0x13 19 0x13 No No No No No No No
<DC4> 20 0x14 20 0x14 No No No No No No No
<NAK> 21 0x15 0 0x00 No No No No No No No
<SYN> 22 0x16 0 0x00 No No No No No No No
<ETB> 23 0x17 0 0x00 No No No No No No No
<CAN> 24 0x18 0 0x00 No No No No No No No
<EM> 25 0x19 0 0x00 No No No No No No No
OK
Cancel
Edit...
Default

**WLinq 3.19 – Character Translation Dialog**

In WLinq 3.x, the only way to have WLinq press special keys like Home, Page Down, and similar was awkward. First you had to choose a character position, then redefine that position to map the character to for example the Home key. Then in the data output format, you had to use that character in the output string, for example: Input() + "\x81".

Freefloat Link has no character translation table, instead you can record a custom key sequence and give it a name:



The key name can then be used as an expression in the string passed to the **send()** method:

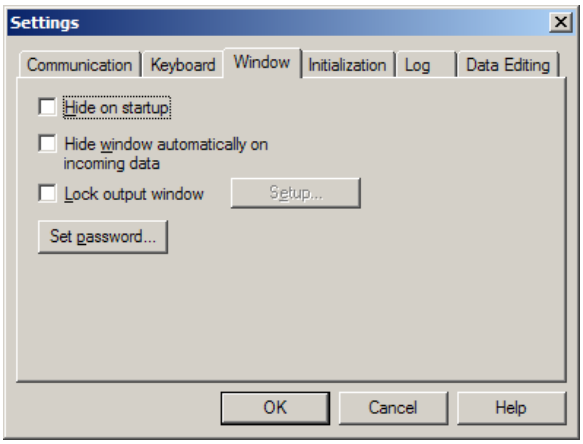
```
function onData( data, length )
  app.send( "{Page Down}" .. data )
end
```

## Send Pre- and Postfix Keys

This feature mainly existed for the integration of WLinq to Freefloat Access*One. When activated, the key sequence Ctrl + Alt + 1 was sent before the data string and Ctrl + Alt + 2 was sent after the data string. It enabled Access*One to distinguish between keyboard and barcode data. To achieve the same result, record the key sequences and given them the names {Prefix} and {Postfix} and then use them in an expression:

```
function onData( data, length )
  app.send( "{Prefix}" .. data .. "{Postfix}" )
end
```

## Lock Output Window

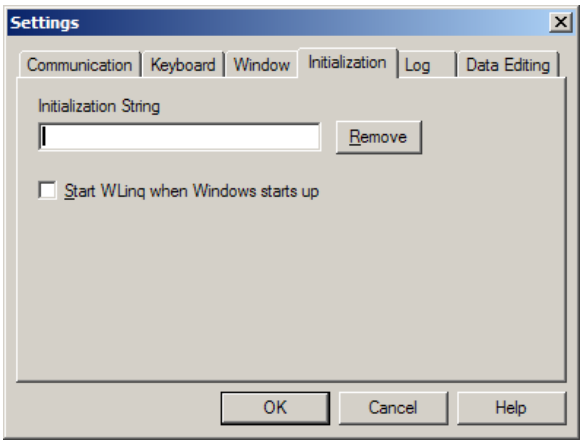



**WLinq 3.19 – Lock Output Window**

The **Lock Output Window** function can be implemented in a script. The following script only sends data to a window who's title contain the text "- Notepad":

```
function onData( data, length )
  windowTitle = app.getWindowText( app.getForegroundWindow() )
  if string.find( windowTitle, "- Notepad" ) then
    app.send( data .. "{Enter}" )
  end
end
```

## Initialization String



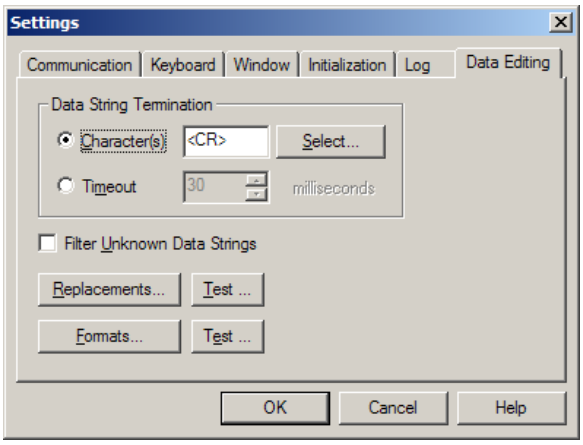

The **Initialization String** can be used for sending a command to for example a barcode scanner that needs some enabling or configuration command at startup.

In Freefloat Link, the following script could instead be used to send commands to the equipment attached to the serial port:

```
function onStart()
  output = "Hello scanner!"
  app.sendSerialData( output, string.len( output ) )
end
```

**Note! onStart()** is called when a profile is activated. This happens when Freefloat Link starts but also when you click **OK** in the **Settings** dialog. Similarly, **onEnd()** is called when Freefloat Link is exited and also when the **Settings** dialog is entered by clicking the **Settings** button in the main window.

## Filter Unknown Data Strings



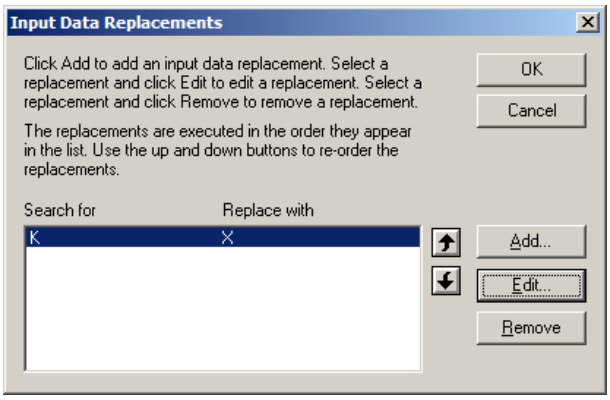

**WLinq 3.19 – Filter Unknown Data Strings**

In WLinq 3.x, if no data editing format matched the input data, the option **Filter Unknown Data Strings** determined whether the input data should be discarded or let through unmodified. The same effect can easily be implemented in a Freefloat Link script:

```
function onData( data, length )
  if string.find( data, "K06", 1, true ) then
    app.send( data .. "{Enter}" )
  end
end
```

The above script makes Freefloat Link filter all input data that doesn't start with the characters K06.

## Input Data Replacements


Input Data Replacements
Click Add to add an input data replacement. Select a replacement and click Edit to edit a replacement. Select a replacement and click Remove to remove a replacement.
The replacements are executed in the order they appear in the list. Use the up and down buttons to re-order the replacements.
OK
Cancel
Search for
Replace with
K
X
Add...
Edit...
Remove

**WLinq 3.19 – Input Data Replacements Dialog**

The replacement function in earlier WLinq versions was quite easy to use. But it lacked power and flexibility. Below is an example of a simple substring replacement. It replaces all occurrences of the character K with the character X.

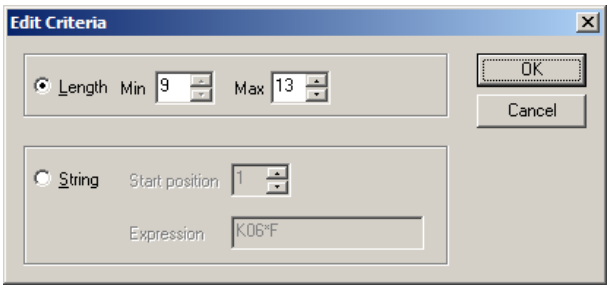
```
function onData( data, length )
  app.send( string.gsub( data, "K", "X" ) .. "{Enter}" )
end
```

Multiple replacements can be done by storing the result in a string variable and repeat the process. Here K is replaced with X and A is replaced with TEST.

```
function onData( data, length )
  local result = string.gsub( data, "K", "X" )

  result = string.gsub( result, "A", "TEST" )

  app.send( result .. "{Enter}" )
end
```

## Criteria

In the previous WLinq generation, a data format was activated for a data string when the criteria of the data format matched. Two types of criteria were supported, length and pattern.



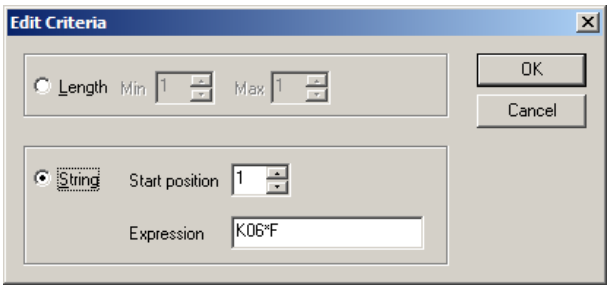

**WLinq 3.19 – Length Criteria for Data Editing Format**

In Freefloat Link the same function as a length criteria is implemented using an if-statement.

```
function onData( data, length )
  if (length >= 9) and (length <= 13) then
    app.send( data .. "{Enter}" )
  end
end
```

An alternative approach could be:

```
function onData( data, length )

  Code modifying the contents of data so that length is no longer valid

  if (string.len( data ) >= 9) and (string.len( data ) <= 13) then
    app.send( data .. "{Enter}" )
  end
end
```




**WLinq 3.19 – Pattern Criteria for Data Editing Format**

A pattern criteria like the one above could be implemented using the **string.find()** pattern matching method:

```
function onData( data, length )
  if string.find( data, "K06.*F" ) then
    app.send( data .. "{Enter}" )
  end
end
```

The format used for patterns in **string.find()** and the format in WLinq 3.x is different. Please refer to the Lua documentation for the Lua pattern format.

A big advantage with scripting in Freefloat Link is that more complex decisions can be made, for example mixing length and pattern matching, something that was not possible in WLinq 3.x.

Multiple criteria used in WLinq 3.x can be implemented by chaining if-elseif-statements:

```
function onData( data, length )
  if length == 9 then
    app.send( "Nine characters: " .. data .. "{Enter}" )
  elseif length == 13 then
    app.send( "Thirteen characters: " .. data .. "{Enter}" )
  else
    app.send( "Not 9 and not 13: " .. data .. "{Enter}" )
  end
end
```

## Data Format Output

In a WLinq 3.x data format, expressions was entered into the data format output edit box and combined with plus (+). In Freefloat Link, all the string operations are using the facilities of the embedded script language. To make Freefloat Link simulate a possibly modified string as keyboard output, you need to pass the string to the method **app.send()**.

Use the table below as a guide for converting expressions in WLinq 3.x to Freefloat Link. Most string operations in WLinq 3.x operated on the data input string implicitly. In Freefloat Link, the data string is an argument sent to the script methods **onData(), onKeyboardCapture()**, and **onExternalData()**.

| **Constant string** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| "ABC" | "ABC" |

| **Extract a substring from the start of the string** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Left( 3 ) | string.sub( data, 1, 3 ) |

| **Extract a substring from the end of the string** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Right( 3 ) | string.sub( data, -3 ) |

| **Extract characters from position three up to position four.<br>Please note the difference in parameters!** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Mid( 3, 2 ) | string.sub( data, 3, 4 ) |

| **From the first A in the string, extract five characters including the A** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Mid( "A", 5 ) | string.gsub( data, ".*(A....).*", "%1" ) |

| **Extracts characters from position six to the end of the string** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Mid( 6 ) | string.sub( data, 6 ) |

| **Scans for the first string and extracts all characters up to the second string.<br>23 and CD is not included in the result.** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| SubStr( "23", "CD" ) | string.gsub( data, ".*23(.*)CD.*", "%1" ) |

| The entire data string | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Input() | data |

| Inserts the current date in the specified format | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Date( "%Y-%m-%d" ) | os.date( "%Y-%m-%d" ) |

| Inserts the current time in the specified format | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Time( "%H:%M" ) | os.date( "%H:%M" ) |

| Concatenations of expressions | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| "X" + Left( 2 ) + Right( 2 ) | "X" .. string.sub( data, 1, 2 ) .. string.sub( data, -2 ) |

| Control characters | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| "<cr><tab>" | "{13}{9}" |
| "\x09" | "{9}" |
| "\d013" | "{13} |

| Combining text and key presses | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Input() + "<tab>1<cr>" | data .. "{Tab}1{Enter}" |

| Reboot Windows | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| Reboot() | app.exitWindows( exitWindowsOpts["REBOOT"] ) |

| Reboot Windows (forced) | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| RebootForced() | app.exitWindows( exitWindowsOpts["FORCE"] ) |

| Starts the specified program | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| RunApp( "notepad.exe" ) | h = app.run( "notepad.exe" )<br>closeAppHandle( h ) |

Please note that the Freefloat Link sample code below more realistically demonstrates what is needed when switching to another application. A small delay is needed before sending input to the activated window or characters may be lost. Also the example avoids an unnecessary delay when the target window already is the foreground window.

| **Activates the first window that has a caption that matches the window caption pattern** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| SetFocus( "*Notepad" ) | function setForegroundWindow( pattern )<br>  local t = app.enumWindows( 0 )<br>  for k,v in pairs(t) do<br>  local title = app.getWindowText(v)<br>  if string.match( title, pattern ) then<br>  app.setForegroundWindow( v )<br>  return<br>  end<br>  end<br>end<br><br>function onData( data, length )<br>  local pattern = ".*Notepad"<br>  local curWindow = app.getForegroundWindow()<br>  local title = app.getWindowText( curWindow )<br>  if not string.match( title, pattern ) then<br>  setForegroundWindow( pattern )<br>  app.sleep( 250 )<br>  end<br>  app.send( data .. "{Enter}" )<br>end |

There is no direct equivalent function for the WLinq 3.x WaitForWindow. Below is a full example of a script which waits for a Notepad window to appear, activates the window, and after a small delay sends the data to the window.

Some common situations where you need to wait for a window are when waiting for an **Open** dialog to appear (after sending Ctrl+O) or when you have launched an application with **app.run()** and need to wait for it to be ready to receive input.

| **Wait for a window to appear** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| WaitForWindow( "*Notepad", 3000 ) | function waitForWindow( pattern, waittime )<br>  local maxwaittime = app.getTickCount() + waittime<br>  local found = false<br><br>  while (app.getTickCount() < maxwaittime) and not found do<br>  local t = app.enumWindows( 0 )<br><br>  for k,v in pairs(t) do<br>  if string.match( app.getWindowText( v ), pattern ) then<br>  found = true<br>  break<br>  end<br>  end<br>  end<br><br>  return found<br>end<br><br>function onData( data, length )<br>  local pattern = ".*Notepad"<br>  if waitForWindow( pattern, 5000 ) then<br>  setForegroundWindow( pattern )<br>  app.sleep( 250 )<br>  app.send( data .. "{Enter}" )<br>  end<br>end |

No direct equivalent function for WaitForAppExit() exists in Freefloat Link. The same result can be achieved by using **app.isAppRunning()**.

Even though the sample below demonstrates a script that pauses until you exits Notepad, Freefloat Link is not intended to have a script that interact with the user (except for **app.messageBox()**) since there may be side effects.

**app.isAppRunning()** is intended to be used to synchronize the script with an external application that does its job and then exits.

**Waiting for an application to exit**

| **WLinq 3.x** | **Freefloat Link** |
|---|---|
| WaitForAppExit( 30000 ) | function onData( data, length )<br>-- Launch Notepad, if it could not be started, the script is aborted<br>-- which means there is no need to check the handle<br>local appHandle = app.run( "notepad.exe" )<br><br>-- Wait until Notepad is exited<br>while app.isAppRunning( appHandle ) do<br>app.sleep( 100 )<br>end<br><br>-- Close the handle to avoid leaks<br>app.closeAppHandle( appHandle )<br><br>-- Tell the user we are done<br>app.messageBox( "Freefloat Link", "Notepad is gone!" )<br>end |

If a script calls **app.closePort()**, the script can start an external application that uses the same serial port. When that external application is exited, the script can re-open the serial port by calling **app.openPort()**.

| **Open the serial port** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| OpenPort() | app.openPort() |

| **Close the serial port** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| ClosePort() | app.closePort() |

| **Send data to the serial port** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| SendData( "abc" ) | data = "abc"<br>app.sendSerialData( data, data:len() ) |

**Note!** The profile switch is performed when the script has finished its execution.

| **Switch profile** | |
|---|---|
| **WLinq 3.x** | **Freefloat Link** |
| SetProfile( "Profile2" ) | app.setProfile( "Profile2" ) |

## Appendix A, Support for Thin Clients, Java Applications, and Flash Applications

Normally, Freefloat Link uses a Windows API function called SendInput to simulate key presses to the active application. This API is recommended by Microsoft because it takes care of differences between different keyboard locales. For example, on a French keyboard, the letter A is positioned where the letter Q is on a US/UK keyboard layout.

However, this technique of simulating keys doesn't work with all environments and applications used on the Windows platform. So far, problems have been spotted with thin clients (Terminal Services or Citrix), Java applications, and Flash applications.

To address this issue, key sequences for digits, lower case and upper case letters has been recorded and is present in the default configuration of Freefloat Link. Script functions for translating digits and letters to key sequences are provided in extra.lua. Also, the function sendData in the default script.txt contains information about how to activate this feature.

Please note that the key sequences are tailored for the most common keyboard layouts, QWERTY with non-shifted keys for digits. You need to modify some of these key sequences to make it work on for example AZERTY (French) and QWERTZ (German) keyboard layouts.

## Appendix B. Lua Copyright

Copyright © 1994-2008 Lua.org, PUC-Rio.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Appendix C. Version History

| Version | Changes | Date |
|---|---|---|
| 5.0 | Major upgrade, product name changed, and OEM version created. | 2008-11-26 |
| 5.1 | Changed name of Lua DLL file to make it compatible with a Lua addon.<br><br>Fixed problem with empty hot key sequences. | 2009-03-06 |
| 5.2 | Added the wizard.<br><br>Updates to the manual. | 2009-04-02 |
| 5.3 | When "{65}" was used as a character notation there was a bug in the parser that sometimes made the character disappear.<br><br>On slow systems with a single CPU core there could be a race condition between the main program module and the licenser module. | 2009-10-29 |
| 5.4 | Added 30-day trial period.<br><br>Added scan codes for all default key sequence definitions. This was done to avoid problems sending Tab and similar keys in a thin-client environment. | 2010-10-05 |
| 5.5 | OEM version released for testing. | 2010-12-22 |
| 5.6 | OEM version with extra tab in Settings dialog for easier configuration of scanner key. | 2011-05-05 |
| 5.7 | Now blocks key sequences named {numbers} since they won't work. The reason is that the syntax {number} is used as a format to specify the ASCII/Unicode code for characters in the string sent to app.send().<br><br>Added built-in support for applications/environments not supporting the way Freefloat Link simulates most keys. Added appendix that explains how this feature is activated and how it works. | 2011-05-10 |
| 5.8 | Removed remnants of code for overlapped I/O that wasn't used. It made third-party serial drivers upset and caused Freefloat Link to hang. This problem was noticed when trying to use Freefloat Link together with BlueSoleil Bluetooth software. | 2011-07-08 |
| 5.9 | Added network communication (Freefloat Datastar). | 2012-01-19 |
| 5.10 | Minor fixes related to network communication. | 2012-02-24 |

| | | |
|---|---|---|
| 5.11 | Fixed a bug that could crash Freefloat Link when data arrived. | 2012-06-07 |
| 5.12 | Fixed a timing problem with trigger keys that crashed the OEM scanner driver. | 2012-07-12 |
| 6.0 | New product name, Freefloat Link. | 2012-08-31 |